МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН

КАЗАХСТАНСКО-НЕМЕЦКИЙ УНИВЕРСИТЕТ
Факультет инжиниринга и информационных технологий

УДК: «007»

НАУЧНАЯ РАБОТА

на тему: «**Использование методов функционального программирования в контексте формирования кнопочных структур»**

Выполнили:
студенты 2-го курса,
Коваленко П. С.
(kovalenkoprohor@mail.ru)
Кравцов М. А.
(maximu17@mail.ru)
Научный руководитель,
Карякин В. П.

Алматы, 2020

Аннотация

В данной работе представлена информация об использовании методов функционального программирования в контексте формирования кнопочных структур. В процессе рассмотрения данного вопроса, были рассмотрены различные варианты избегания коллизий, характерных для разработки приложений, содержащих кнопочные структуры, связанных с особенностями языка программирования «Python», такими как: ссылки, а также низкая скорость генерации графических объектов. Рассмотрение данного вопроса производится в рамках написания ряда программ. К примеру, клеточный автомат – «Жизнь», разработанный английским математиком Джоном Конвеем в 1970 году, является отличным вариантом для изучения и практики в области формирования кнопочных структур. Основное рассмотрение производится при помощи стандартной библиотеки языка «Python» - «Tkinter», а также реализация осуществляется и при взаимодействии с другими библиотеками, такими как:

- «Pygame»
- «Pytube»
- «re»
- «ttk»

Данная проблема актуальна, когда речь заходит об оптимизации процесса разработки приложений, предназначенных, как для мобильных устройств, так и для персональных компьютеров. С многими из рассматриваемых задач нам пришлось столкнуться при устройстве на работу, и при более углубленном изучении возможностей языка «Python» в области разработки.

Ключевые слова.

- Кнопка
- Коллизия
- Генератор
- Функция
- «Python»
- Проблема
- Ссылки
- Структура данных

Оглавление

ВВЕДЕНИЕ

Часто начинающие разработчики сталкиваются с процессом программирования множества однотипных объектов в своём приложении. Самым простым примером является программирование клавиатуры для системы Android. Казалось бы, в чём проблема – берёшь и программируешь объекты, размещаешь их и добавляешь свой функционал. Но данный метод чреват огромными программными кодами и большим количеством однотипных функций. Особенно это касается объектно-ориентированной парадигмы программирования, которая и так характеризуется малой компактностью программ. По данным причинам многие сразу бегут на «Habr» в целях найти готовые блоки или используют множество встроенных методов, чтобы избежать написания километровых кодов.

ОСНОВНАЯ ЧАСТЬ

Обоснование выбранного языка программирования

Мы предлагаем свой вариант решения данной проблемы конкретно для языка программирования «Python» – использовать методы функционального программирования для формирования удобного и рабочего интерфейса. Выбранный язык в данный момент довольно популярен в области разработки кроссплатформенных приложений по причине своей мультипарадигмийности и удобства работы с формами и их объектами. Для приведения примера стандартного метода написания какой-либо формы и её объектов мы будем использовать встроенную графическую библиотеку «tkinter». Она позволяет пользоваться готовыми стандартными элементами, такими как кнопка и метка для формирования сложных структур вроде клавиатуры или гистограммы, что позволяет избавить нас от необходимости описывать длинный процесс создания какого-либо объекта с чистого листа, в процессе демонстрации нашей идеи. Теперь стоит переключиться непосредственно в компилятор Python’a (рис. 1):

```
from tkinter import*#привязываю библиотеку
tk=Tk()#создаю форму
tk.geometry("600x480")#задаю размер формы
b=Button(tk,text='Кнопка')#создаю кнопку
b.pack()#активирую отображение кнопки
b.place(x=0,y=0,width=120,height=120)#размещаю кнопку на экране
```

Рис. 1

Пример работы кода выше:

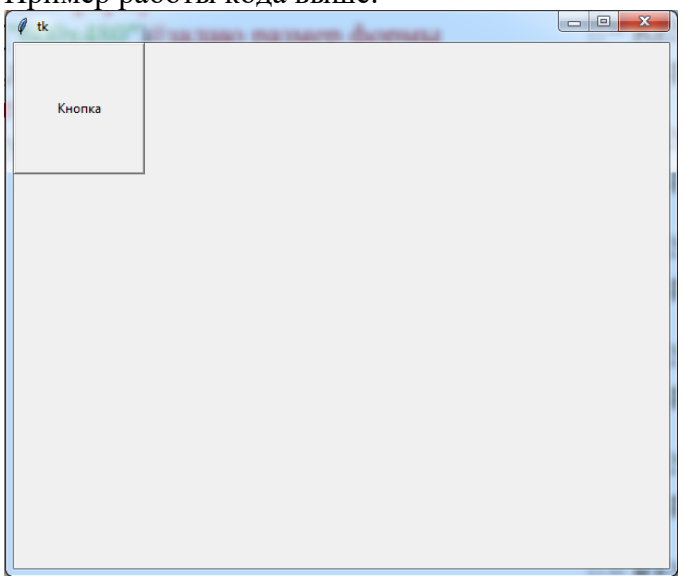

Рис. 2

В данной программе мы видим, что кнопка сформировалась и отображается на экране. Однако кнопка не имеет никакой смысловой нагрузки покуда у неё нет функционала. Добавим ей какую-нибудь функцию, которая будет вызываться при нажатии на кнопку (рис.3):

```
from tkinter import*#привязываю библиотеку

tk=Tk()#создаю форму
tk.geometry("600x480")#задаю размер формы

def click():#функция клика по кнопке
    b['text']='Кнопка нажата'#изменения текста кнопки

b=Button(tk,text='Кнопка',command=click)#создаю кнопку
b.pack()#активирую отображение кнопки
b.place(x=0,y=0,width=120,height=120)#размещаю кнопку на экране
```

Рис. 3

Результат, после нажатия на кнопку (рис. 4):

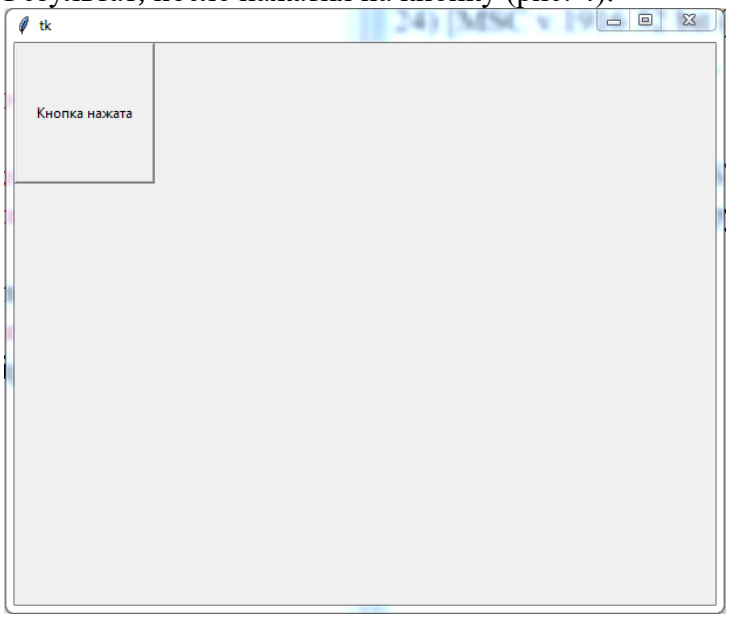

Рис. 4

Описание проблем создания кнопочной структуры

С созданием одной кнопки проблем не возникает. Теперь попробуем создать хотя бы 5 штук с разными назначениями. После окончания 11 класса, одному из нас поручили проект, где суть задания состояла в том, что необходимо было с нескольких устройств проголосовать за что-либо по пятибалльной шкале, и вся информация собиралась на независимом устройстве. Абстрагируясь от создания локальной сети, попробуем повторить тоже самое в «tkinter’е» в Python (рис. 5):

```
from tkinter import*#привязываю библиотеку
tk=Tk()#создаю форму
tk.geometry("600x480")#задаю размер формы

def click1():#функция клика по кнопке
    b1['bg']='#000000'#изменения цвета кнопки
    b1['fg']='#ffffff'#изменения цвета текста кнопки

def click2():#функция клика по кнопке
    b2['bg']='#6a422d'#изменения цвета кнопки
    b2['fg']='#ffffff'#изменения цвета текста кнопки
def click3():#функция клика по кнопке
    b3['bg']='#ff0000'#изменения цвета кнопки
    b3['fg']='#ffffff'#изменения цвета текста кнопки
def click4():#функция клика по кнопке
    b4['bg']='#ffff00'#изменения цвета кнопки
def click5():#функция клика по кнопке
    b5['bg']='#00ff00'#изменения цвета кнопки

b1=Button(tk,text='1',font='TimesNewRoman 20',command=click1)#создаю кнопку
b1.pack()#активирую отображение кнопки
b1.place(x=0,y=0,width=120,height=120)#размещаю кнопку на экране

b2=Button(tk,text='2',font='TimesNewRoman 20',command=click2)#создаю кнопку
b2.pack()#активирую отображение кнопки
b2.place(x=120,y=0,width=120,height=120)#размещаю кнопку на экране

b3=Button(tk,text='3',font='TimesNewRoman 20',command=click3)#создаю кнопку
b3.pack()#активирую отображение кнопки
b3.place(x=240,y=0,width=120,height=120)#размещаю кнопку на экране

b4=Button(tk,text='4',font='TimesNewRoman 20',command=click4)#создаю кнопку
b4.pack()#активирую отображение кнопки
b4.place(x=360,y=0,width=120,height=120)#размещаю кнопку на экране

b5=Button(tk,text='5',font='TimesNewRoman 20',command=click5)#создаю кнопку
b5.pack()#активирую отображение кнопки
b5.place(x=480,y=0,width=120,height=120)#размещаю кнопку на экране
```

Рис. 5

Взгляд на форму перед нажатием на кнопки (рис. 6):

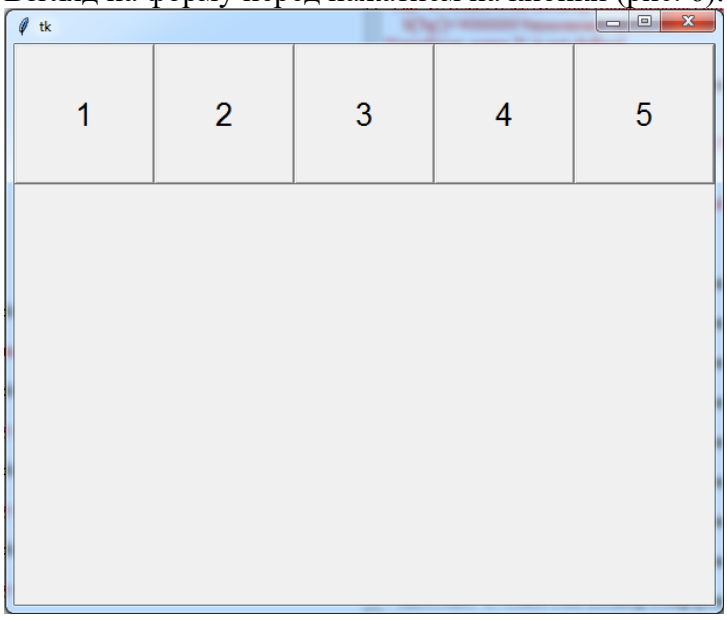

Рис. 6

Взгляд на форму после нажатия на кнопки (рис. 7):

Рис. 7

Несложно заметить, что всё работает корректно, но, взглянув на размер программного кода, чувства возникают смешанные. Здесь количество строк линейно зависит от количества кнопок. Объяснение этому можно найти в коде, где добавлялся функционал для одной кнопки. Для одной кнопки необходимо три строки для её формирования, прорисовки и расположения, а также функция, вызываемая при клике. Из этого и складывается неприличных размеров код. Однако, не стоит быть особо внимательным человеком, чтобы заметить, что большинство строк главного метода циклически идентичны, что даёт надежду на то, что всё это можно разместить в теле цикла и, на основе этого, формировать кнопки. Почему так не делают?

- Первая и основная проблема – запись функции клика кнопки. Если попробовать циклически формировать кнопки, то циклически придётся формировать и функции, вызываемые при клике на кнопку. Казалось бы, можно просто задавать аргументы функции и, в зависимости от них, записывать функционал кнопки. Но проблема в том, что функцию, которая запускается при активации кнопки нельзя вызвать от определённого аргумента. Здесь ситуация схожа с такими функциональными структурами как «map» и «filter» – функция указывается в аргументах, но не вызывается.
- Вторая проблема – выбор структуры данных для записи совокупности кнопок. Структура должна быть изменяемой и простой в использовании, так как в функции клика частенько придётся обращаться к той или иной кнопке, а именно к её аргументам, таким как текст кнопки или цвет фона. Обычно для этих целей используют «array» из одноимённой библиотеки, однако мы являемся большими фанатами динамических структур, и зачастую используем списки (в том числе и вложенные – просто и со вкусом).

Рассмотрение коллизии о структуре данных

Для начала лучше решать вторую проблему. Она менее объёмная и всё, что нужно знать для её решения – это то, что все операторы, функции, переменные в «Python» являются объектами каких-либо классов, и, следовательно, по аналогии с объектно-ориентированной парадигмой любого языка, из объектов можно создать массив, а в нашем случае список. Приведу несложные примеры генерации списков из элементов класса «int», «None» и «builtin_function» (рис. 8):

```
>>> [i for i in range(10)]#генерация списка из целых чисел
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> [print(i) for i in range(10)]#генерация списка из выводов
0
1
2
3
4
5
6
7
8
9
[None, None, None, None, None, None, None, None, None, None]
>>> a=[int for i in range(10)]#генерация списка из функций
>>> a[0]('678')
678
```

Рис. 8

Точно таким-же образом можно сгенерировать список кнопок. Но, помимо генерации кнопок, нужно также добавить их прорисовку и расстановку. Для этого тоже можно использовать генераторы. Посмотрим, как преобразится код с 5-ю кнопками (рис. 9):

```
from tkinter import*#привязываю библиотеку
tk=Tk()#создаю форму
tk.geometry("600x480")#задаю размер формы

b=[Button(tk,text=str(i+1),font='TimesNewRoman 20')for i in range(5)]#генерация кнопок
[b[i].pack()for i in range(5)]#генерация отображений кнопок
[b[i].place(x=0+120*i,y=0,height=120,width=120)for i in range(5)]#генерация расположений
```

Рис. 9

Вид на форму (рис. 10):

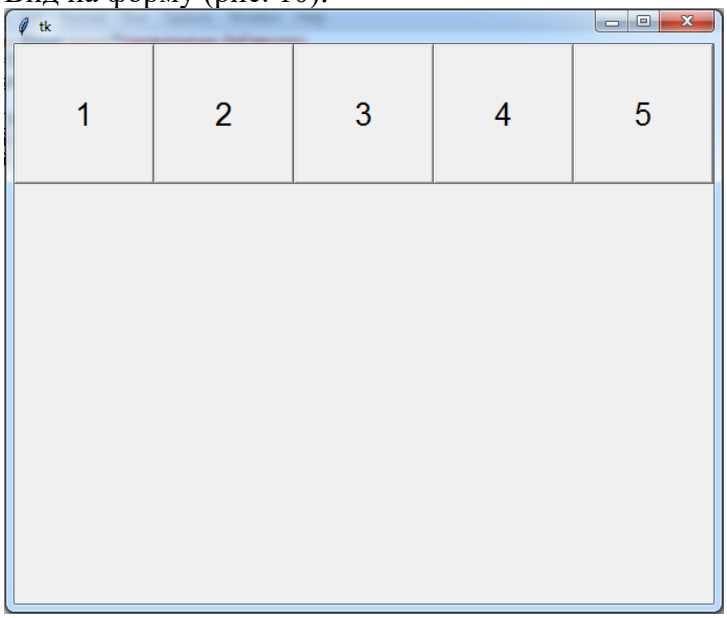

Рис. 10

Уверены, что невооружённым глазом стало заметно, как уменьшился и стал понятнее код программы. Но тут присутствуют три генератора с одинаковыми параметрами оператора

перебора переменной «for». Это стоит исправить – оставить общий «for» для всех трёх операций, а сами операции поместить в блок тела цикла (рис. 11):

```
from tkinter import*#привязываю библиотеку
tk=Tk()#создаю форму
tk.geometry("600x480")#задаю размер формы
b=[]
for i in range(5):
    b.append(Button(tk,text=str(i+1),font='TimesNewRoman 20'))#создание кнопки
    b[i].pack()#отображение кнопки
    b[i].place(x=0+120*i,y=0,height=120,width=120)#расположение кнопки
```

Рис. 11

Рассмотрение вариантов реализации функционала кнопочной структуры

В целом работа программы не меняется, но незначительно ускоряется процесс формирования кнопок. Но проблема и того, и другого варианта в том, что у кнопок нет функционала. Какие есть варианты его добавить?

1. Использовать лямбда-функцию, генерирующуюся прямо в процессе создания кнопок в аргументах кнопки. Такой способ плох тем, что многие идеи сложно воплотить в рамках лямбда функции. Код получится длинный и непонятный, ведь, даже в одной строке лямбда-функции легко заблудиться.
2. Использовать заранее заготовленные функции, которые будут находиться в отдельном списке.
3. Использовать замыкание. То есть, создавать лямбда-функцию, которая будет вызывать другую функцию относительно какого-либо аргумента.

Мы продемонстрируем все 3 варианта и увидим разницу в работе и сопутствующие подводные камни, характерные для «Python». Для удобства дадим кнопкам схожий функционал – при нажатии на кнопку происходит смена имени формы на номер кнопки.

Первый вариант. Генерация лямбда-функции в аргументах кнопки

```
from tkinter import*#привязываю библиотеку
tk=Tk()#создаю форму
tk.geometry("600x480")#задаю размер формы
b=[]
for i in range(5):
    b.append(Button(tk,text=str(i+1),font='TimesNewRoman 20',command=lambda:tk.title(str(i+1))))#создание кнопки
    b[i].pack()#отображение кнопки
    b[i].place(x=0+120*i,y=0,height=120,width=120)#расположение кнопки
```

Рис. 12

Внешне форма ничуть не изменилась (рис. 13):

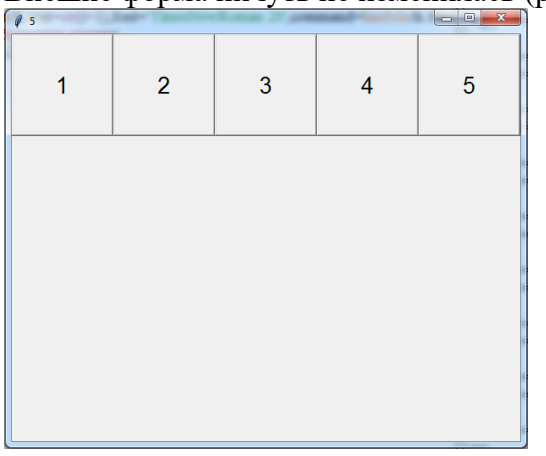

Рис. 13

Рассмотрение коллизии о ссылочной записи

Однако, при нажатии на любую из кнопок результат один – название окна меняется на пятёрку. Почему так происходит? В языке «Python» присутствует интересная система сохранения переменных и часто вместо записи данных записываются ссылки на идентичные данные. В таких случаях при изменении одного объекта меняется другой. Отличным примером может послужить генерация списка посредством дублирования другого списка (рис. 14):

```
>>> a=[1,1]
>>> b=[a]*5
>>> b
[[1, 1], [1, 1], [1, 1], [1, 1], [1, 1]]
>>> b[0][0]=2
>>> b
[[2, 1], [2, 1], [2, 1], [2, 1], [2, 1]]
```

Рис. 14

Из примера можно увидеть, что при изменении одного из элементов вложенного списка меняются и все остальные. Это связано с тем, что вместо значения списка «a», дублируется ссылка на него. Возможно, звучит довольно запутанно, поэтому необходимо привести ещё один пример уже касательно функций (рис. 15):

```
>>> f=[lambda:print(i)for i in range(10)]
>>> f[4]()
9
>>> f[3]()
9
>>> f[7]()
9
```

Рис. 15

Принцип работы ссылок, кажется, понятен, но проблема от этого не решается. Как побороть ссылки? Честно сказать, на данный момент нам удалось найти только один метод – прогнать переменную, которая записывается в виде ссылки через лямбда-функцию типа «y=f(x)», где в роли f(x) будет тот объект, который мы хотим генерировать. Сразу стоит продемонстрировать это на последнем примере, так как он наиболее актуален для нашей задачи (рис. 16):

```
>>> f=[(lambda x:lambda:print(x))(i)for i in range(10)]
>>> f[4]()
4
>>> f[3]()
3
>>> f[7]()
7
```

Рис. 16

Несмотря на то, что код стал немного длиннее, на быстродействии работы функции это не отражается. Лямбда формируется в процессе работы генератора только один раз, а в последствии вызывается уже записанная в память функция. Применим теперь этот метод уничтожения ссылок на нашей форме (рис. 17):

```
from tkinter import*#привязываю библиотеку
tk=Tk()#создаю форму
tk.geometry("600x480")#задаю размер формы
b=[]
for i in range(5):
    #создание кнопки
    b.append(Button(tk,text=str(i+1),font='TimesNewRoman 20',command=(lambda x:lambda:tk.title(str(x)))(i+1)))
    b[i].pack()#отображение кнопки
    b[i].place(x=0+120*i,y=0,height=120,width=120)#расположение кнопки
```

Рис. 17

Теперь кнопки работают корректно и изменяют название формы на свою цифру (рис. 18):

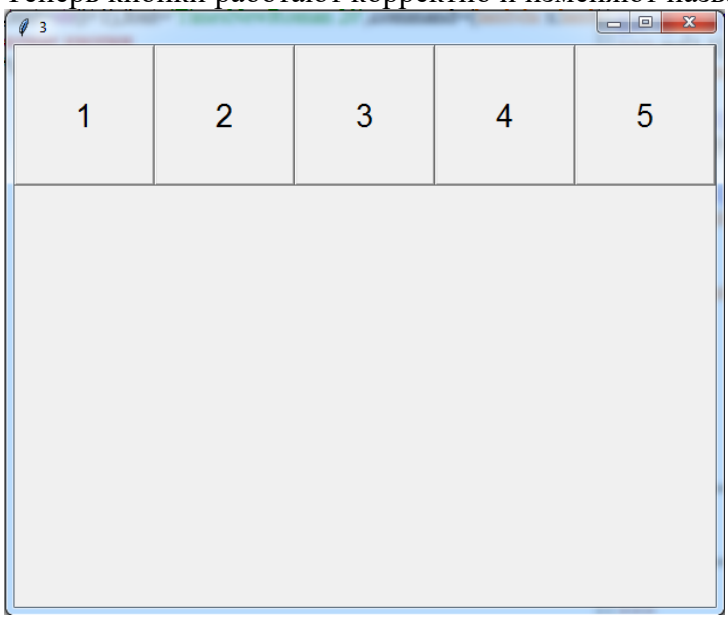

Рис. 18

Второй вариант. Использование заранее заготовленных функций

В данном методе можно использовать как функции, построенные на принципах лямбда-исчисления, так и стандартные функции «def». Так как лямбда-функции ограничивают нас в действиях. Используем «def» с какими-нибудь аргументами для удобства. Однако, подобный вариант подразумевает вызов функции во время генерации списка функций. Вызывать функцию до клика на кнопку нам не нужно, поэтому нам всё же придётся обратиться к лямбда-исчислению для сохранения «вызываемости» (callable) объектов списка (рис. 19):

```
from tkinter import*#привязка библиотеки
tk=Tk()#создание формы
tk.geometry("600x480")#задаю размер формы
def click(i):#создание функции
    tk.title(str(i))
b=[]
f=[lambda:click(i+1)for i in range(5)]#формирование списка функций
for i in range(5):
    #создание кнопки
    b.append(Button(tk,text=str(i+1),font='TimesNewRoman 20',command=f[i]))
    b[i].pack()#отображение кнопки
    b[i].place(x=0+120*i,y=0,height=120,width=120)#расположение кнопки
```

Рис. 19

Вид на форму (рис. 20):

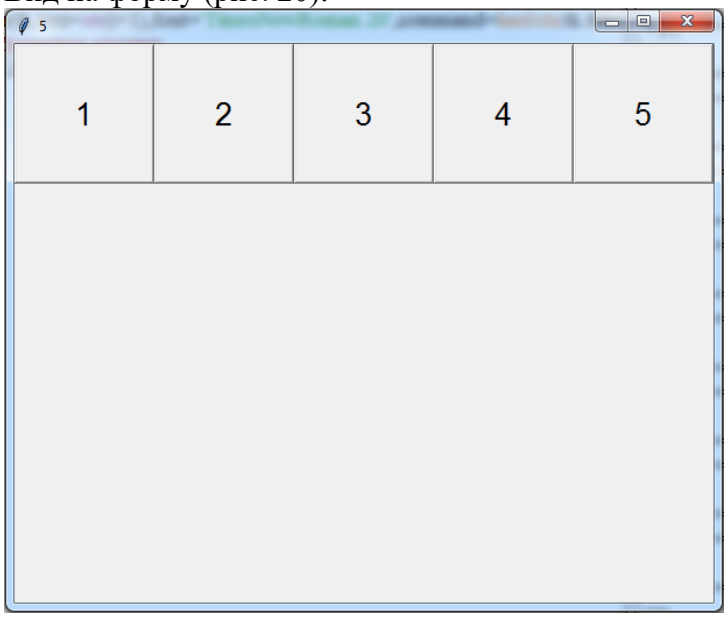

Рис. 20

Здесь нам приходится столкнуться с уже знакомой проблемой. Ссылки не позволяют создать различный функционал для кнопок, и любая кнопка формирует пятёрку в строку заголовка формы. Исправляем (рис. 21):

```
from tkinter import*#привязка библиотеки
tk=Tk()#создание формы
tk.geometry("600x480")#задаю размер формы
def click(i):#создание функции
    tk.title(str(i))
b=[]
f=[lambda:click(i+1)for i in range(5)]#формирование списка функций
for i in range(5):
    #создание кнопки
    b.append(Button(tk,text=str(i+1),font='TimesNewRoman 20',command=(lambda x:f[x])(i)))
    b[i].pack()#отображение кнопки
    b[i].place(x=0+120*i,y=0,height=120,width=120)#расположение кнопки
```

Рис. 21

Однако, при запуске программы обнаруживается, что ссылки до сих пор остались, и кнопки как прежде не работают. Это связано с тем, что ссылки появляются ещё на стадии формирования списка функций. Уберем их оттуда тоже (рис. 22):

```
from tkinter import*#привязка библиотеки
tk=Tk()#создание формы
tk.geometry("600x480")#задаю размер формы
def click(i):#создание функции
    tk.title(str(i))
b=[]
f=[(lambda x:lambda:click(x))(i+1)for i in range(5)]#формирование списка функций
for i in range(5):
    #создание кнопки
    b.append(Button(tk,text=str(i+1),font='TimesNewRoman 20',command=(lambda x:f[x])(i)))
    b[i].pack()#отображение кнопки
    b[i].place(x=0+120*i,y=0,height=120,width=120)#расположение кнопки
```

Рис. 22

Теперь программа работает корректно (рис. 23):

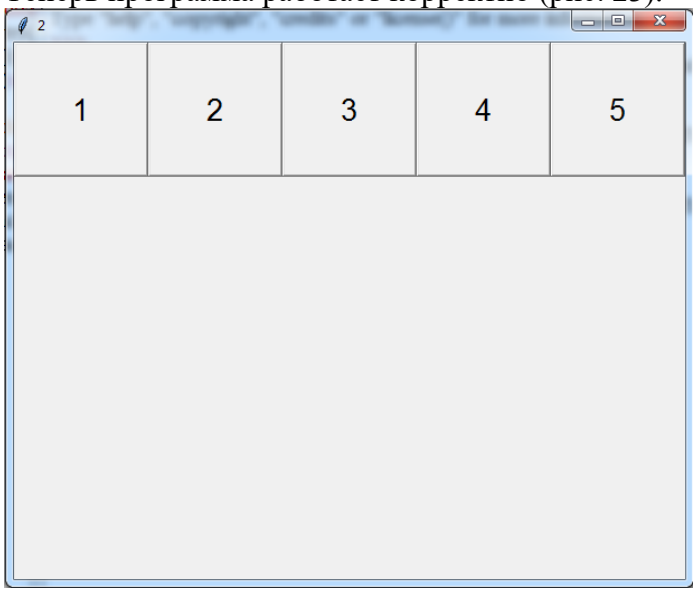

Рис. 23

Второй способ плох прежде всего тем, что в двух местах формируются ссылки и избавляться от них становится ещё более проблематично.

Третий вариант. Использование замыкания

Данный способ подразумевает вызов функции по аргументу прямо в кнопке, занося вызываемую функцию в лямбда-функцию для корректной работы кнопки и сохранения параметра «вызываемый» (callable). Попутно избавившись от ссылок, можно получить подобный код (рис. 24):

```
from tkinter import*#привязка библиотеки
tk=Tk()#создание формы
tk.geometry("600x480")#задаю размер формы
def click(i):#создание функции
    tk.title(str(i+1))
b=[]
for i in range(5):
    #создание кнопки
    b.append(Button(tk,text=str(i+1),font='TimesNewRoman 20',command=(lambda x:lambda:click(x))(i)))
    b[i].pack()#отображение кнопки
    b[i].place(x=0+120*i,y=0,height=120,width=120)#расположение кнопки
```

Рис. 24

Программа работает идентично со всеми предыдущими способами: название окна меняется в зависимости от нажатой кнопки.

Подведение итогов

Теперь самое время попробовать переписать программу, которая меняла цвета кнопочек, с помощью функционального программирования (рис. 25):

```
from tkinter import*
tk=Tk()
tk.geometry("600x480")
bg=['#000000','#6a422d','#ff0000','#ffff00','#00ff00']
def click(i):
    b[i]['bg']=bg[i]
    b[i]['fg']='#000000' if i>2 else '#ffffff'
b=[]
for i in range(5):
    b.append(Button(tk,text=str(i+1),font='TimesNewRoman 20',command=(lambda x:lambda:click(x))(i)))
    b[i].pack()
    b[i].place(x=0+120*i,y=0,width=120,height=120)
```

Рис. 25

Данный код будет весьма полезно сравнить с его первоначальной версией, чтобы увидеть превосходство функционального программирования над чистым объектно-ориентированным подходом (рис. 26):

```
from tkinter import*#привязываю библиотеку
tk=Tk()#создаю форму
tk.geometry("600x480")#задаю размер формы

def click1():#функция клика по кнопке
    b1['bg']='#000000'#изменения цвета кнопки
    b1['fg']='#ffffff'#изменения цвета текста кнопки

def click2():#функция клика по кнопке
    b2['bg']='#6a422d'#изменения цвета кнопки
    b2['fg']='#ffffff'#изменения цвета текста кнопки
def click3():#функция клика по кнопке
    b3['bg']='#ff0000'#изменения цвета кнопки
    b3['fg']='#ffffff'#изменения цвета текста кнопки
def click4():#функция клика по кнопке
    b4['bg']='#ffff00'#изменения цвета кнопки
def click5():#функция клика по кнопке
    b5['bg']='#00ff00'#изменения цвета кнопки

b1=Button(tk,text='1',font='TimesNewRoman 20',command=click1)#создаю кнопку
b1.pack()#активирую отображение кнопки
b1.place(x=0,y=0,width=120,height=120)#размещаю кнопку на экране

b2=Button(tk,text='2',font='TimesNewRoman 20',command=click2)#создаю кнопку
b2.pack()#активирую отображение кнопки
b2.place(x=120,y=0,width=120,height=120)#размещаю кнопку на экране

b3=Button(tk,text='3',font='TimesNewRoman 20',command=click3)#создаю кнопку
b3.pack()#активирую отображение кнопки
b3.place(x=240,y=0,width=120,height=120)#размещаю кнопку на экране

b4=Button(tk,text='4',font='TimesNewRoman 20',command=click4)#создаю кнопку
b4.pack()#активирую отображение кнопки
b4.place(x=360,y=0,width=120,height=120)#размещаю кнопку на экране

b5=Button(tk,text='5',font='TimesNewRoman 20',command=click5)#создаю кнопку
b5.pack()#активирую отображение кнопки
b5.place(x=480,y=0,width=120,height=120)#размещаю кнопку на экране
```

Рис. 26

Прикладное значение

Как и во всех областях нашей жизни, некоторые вещи могут быть гладкими только на бумаге. Чтобы подобные сомнения в функциональной оптимизации создания кнопочных структур развеять, стоит привести примеры написанных с помощью данного метода программ.

Арканоид

Если кто не играл в детстве или забыл название, Арканоид – серия игр на ПК и мобильные устройства, где шариком нужно разбивать разные блоки. Самое тривиальное использование кнопочной структуры – создание кирпичной стены для Арканоида (рис. 27):

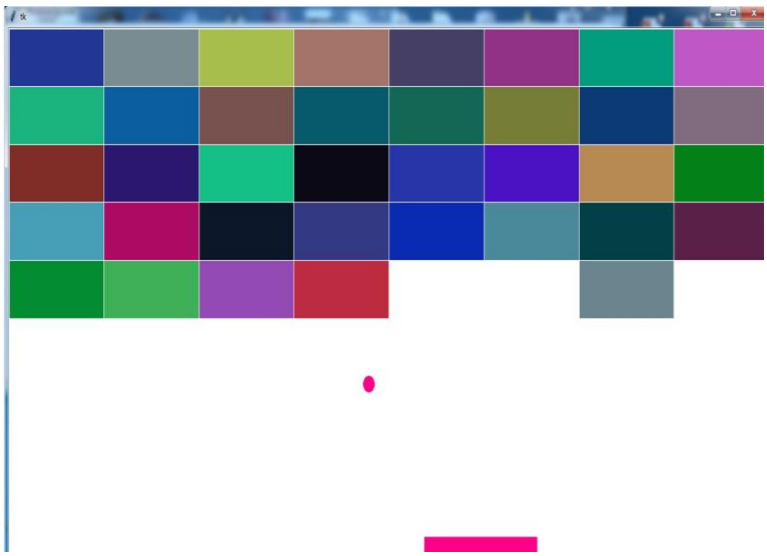

Рис. 27

В случае с Арканоидом пример создания кнопочных структур не подразумевает создание функционала для кнопок, что не полностью отвечает выбранной теме. Тем не менее, даже здесь необходимо было использовать функциональные методы для единичного удаления блоков при попадании шарика.

Игра «Жизнь»

Другим отличным примером реализации создания клеточных структур с помощью функциональной парадигмы языка «Python» является клеточный автомат «Жизнь» 1970-го года. Суть алгоритма в том, что вся плоскость разделена на квадратики, в которых либо есть жизнь (квадратик закрашен), либо нет (квадратик пустой). Если вокруг пустой клетки есть хотя бы 3 закрашенные клетки, то в ней появляется жизнь (аналог размножения). Если вокруг закрашенной клетки больше 3 закрашенных клеток, то клетка снова становится пустой (аналог смерти от перенаселения). Также клетка становится пустой, если вокруг неё меньше двух закрашенных клеток (аналог смерти от скуки).

Несмотря на то, что наша реализация данной игры подразумевает существование и функционирование структуры из 900 кнопок, программа содержит всего на всего 20 строк кода (рис. 28):

```
from tkinter import *
from time import*
def nex():
    old=[uni[i][:]for i in range(size)]
    for y in range(size):
        for x in range(size):
            if old[y][x]['text']=='0':
                if sum([sum(map(lambda x:int(x['text']),i[max(0,x-1):x+2]))for i in old[max(0,y-1):y+2]])==3:b[y][x]()
            else:
                if sum([sum(map(lambda x:int(x['text']),i[max(0,x-1):x+2]))for i in old[max(0,y-1):y+2]])>4:w[y][x]()
                if sum([sum(map(lambda x:int(x['text']),i[max(0,x-1):x+2]))for i in old[max(0,y-1):y+2]])<3:w[y][x]()
o,size,tk,st,uni=0,30,Tk(),(lambda e:nex()if ord(e.char)==13 else None),[]
win=Canvas(tk,width=500,height=550)
win.pack()
b,w=[[(lambda v,u:(lambda:(uni[v].pop(u),uni[v].insert(u,Button(tk,font='Arial 0',fg='#000000',bg='#000000',text='1',command=w[v][u])),uni[v][u].place(relx=u/size,rely=v/(size+1),relwidth=1/(size+1),relheight=1/(size+1)))))(y,x)
uni=[[Button(tk,font='Arial 0',fg='#FFFFFF',bg='#FFFFFF',text='0',command=b[y][x])for x in range(size)]for y in range(size)]
[[uni[y][x].place(relx=x/size,rely=y/(size+1),relwidth=1/(size+1),relheight=1/(size+1))for x in range(size)]for y in range(size)]
Button(tk,font='Arial 20',fg='#FFFFFF',bg='#000000',text='Clear',command=lambda:[[w[y][x]()for x in range(size)]for y in range(size)]).place(relx=0,rely=size/(size+1),relwidth=1/3,relheight=1/(size+1)),Button(tk,font='Arial 20',
win.bind_all("<Key>",st),tk.mainloop()
```

Рис. 28

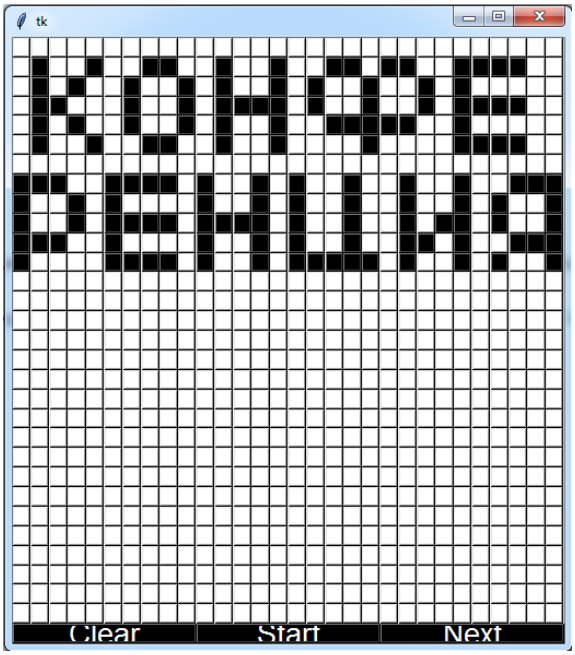

Рис. 29

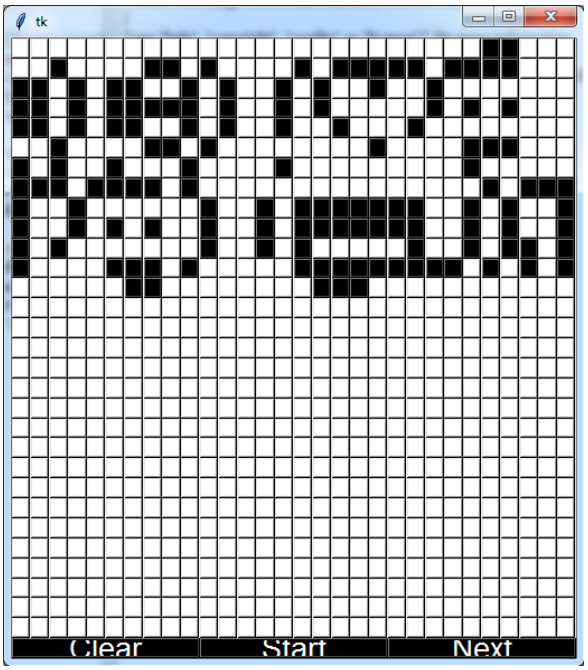

Рис. 30

Калькулятор систем счисления

В самом начале нашего повествования было сказано про приложения на Android. Слово мы своё держим, и можем с уверенностью заявить, что данный метод обработки кнопочных структур также работает и на операционных системах мобильных устройств. В качестве примера приводим калькулятор по переводу числа из одной системы счисления в другую (рис. 31, 32):

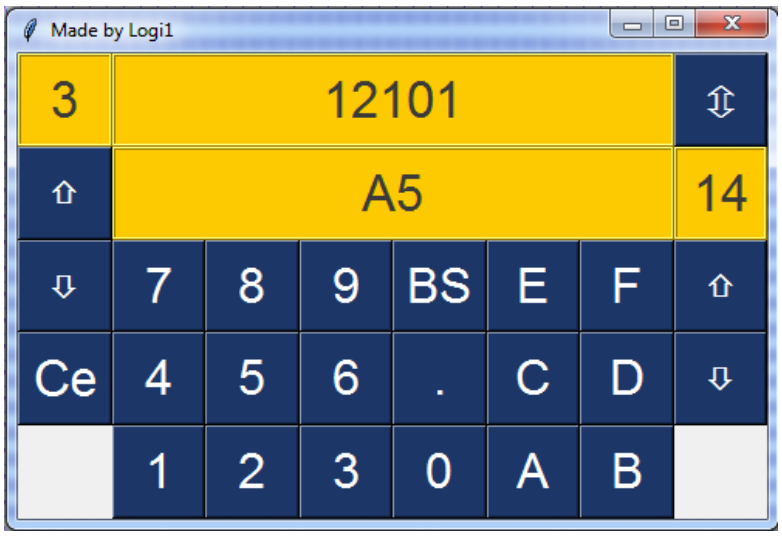

Рис. 31

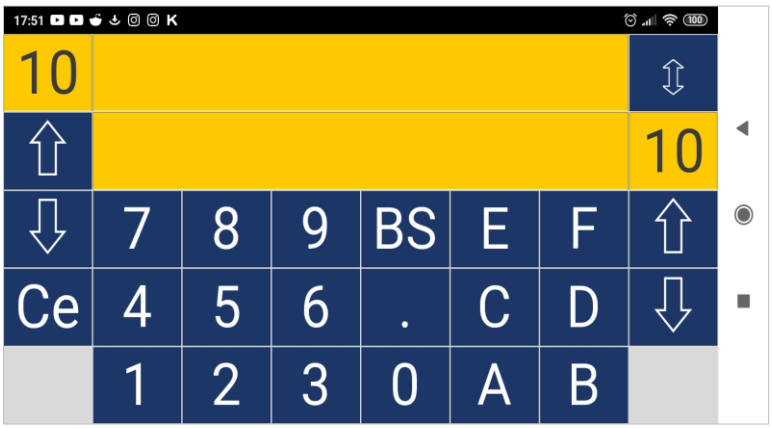

Рис. 32

Данное приложение работает как на устройствах, поддерживающих Windows, так и на Android.

Пример взаимодействия с другими библиотеками

Хотелось бы уточнить, что предложенные методы обработки кнопочных структур не ограничиваются на работе с библиотекой «tkinter». Один из нас отдельно изучил полезные взаимодействия функциональной парадигмы с другими популярными библиотеками, к примеру – «pygame», «pytube». Совместными усилиями нам удалось написать копию популярного во всём мире расширения SaveFrom, которое позволяет скачивать видеоролики с платформы «YouYube», в качестве отдельного приложения. Впоследствии, данное приложение несложно адаптировать и под мобильные устройства, что будет актуально, так как расширения в мобильной версии различных браузеров временами работают некорректно.

Код данной программы (рис. 33):

```
from tkinter import*
from tkinter.ttk import Combobox
from pytube import YouTube
import pygame
pygame.mixer.init()
sound1=pygame.mixer.Sound('Sound.wav') # Загрузка звука

win = Tk()
win.title('SaveFromNet PRO') # Название программы
win.geometry('300x300') # Размер окна

background_image=PhotoImage(file = 'pik.png') # Загрузка фона
background_label = Label(win, image=background_image) # Установка фона
background_label.place(x=0,y=0, relwidth = 1, relheight = 1)

win.bind('<Control-v>') # Возможность использования сочетания ctrl+v

def vidos():
    x = Entry.get() # Получение ссылки на видео
    url = x
    sound1.play() # Воспроизведение звука
    yt = YouTube(url)

    print('Имя видеоролика: ',yt.title)
    print('\nКол-во просмотров: ',yt.views)
    print('\nДлина видео в секундах: ',yt.length)
    print('\nОписание: ',yt.description)
    print('\nРейтинг: ',yt.rating)

    try: # Обработка исключения
        if cmb.get() == '360p':
            yt.streams.filter(res="360p").first().download()
        elif cmb.get() == '480p':
            yt.streams.filter(res="480p").first().download()
        elif cmb.get() == '720p':
            yt.streams.filter(res="720p").first().download()
        else:
            video = yt.streams.get_highest_resolution()
            video.download()
    except AttributeError: # Перехват ошибки
        print('\nРАЗРЕШЕНИЕ НЕ СОВПАДАЕТ')

lb = Label(win, text = 'Ссылка на видео', font = 'Impact 12') # Надпись
lb.place(x = 95, y = 5)

Entry = Entry(win, width = 25, bg = 'red') # Текстовое поле
Entry.place(x = 80, y = 30, height = 30)

cmb = Combobox(win, width = 20, values = ('360p', '480p', '720p', 'high')) # Комбобокс с вариантами разрешения
cmb.place(x = 85, y = 65, height = 30)

tube = Button(win, text = 'DOWNLOAD', bg = 'green', font = 'Impact 12', width = '10', height = '2', fg="white",command=vidos) # Кнопка для скачивания
tube.place(x=110,y=100)

qt = Button(win, text = 'OK!', bg = 'red', font = 'Impact 12', width = '10', height = '2', fg="white", command = win.destroy) # Выход из программы
qt.place(x = 110, y = 150)
win.mainloop()
```

Рис. 33

Помимо предоставления возможности скачивания видеоролика, данная программа предоставляет информацию о: названии видеоролика, количестве просмотров, длине видео в секундах, его описание, а также соотношение лайков и дизлайков.

Кроме этого, пользователь может опционально выбрать разрешение, в котором выбранное видео будет загружено. Для осуществления загрузки, пользователю нужно лишь вставить ссылку с видеороликом в программу, посредством сочетания клавиш «ctrl+v», и выбрать необходимое разрешение видеоролика, после чего нажать на кнопку «download», по окончанию загрузки, нажать на кнопочку «ok», для выхода из программы. Если пользователь по ошибке выбрал разрешение, превышающее разрешение самого видеоролика, то ничего страшного. Программа это обработает, и выдаст пользователю сообщение о том, что разрешение самого видеоролика, и разрешение указанное в программе, не совпадают. Также нажатие кнопки «download», сопровождается звуковым эффектом, полученным благодаря использованию библиотеки «pygame».

Интерфейс программы (рис 34, 35, 36):

Рис. 34

Рис. 35

```
Имя видеоролика:  НАШИ ДЕЙСТВИЯ НАОБОРОТ! / OUR  ACTIONS IN REVERSE!

Кол-во просмотров:  313

Длина видео в секундах:  225

Описание:  Я ДУМАЮ ДАННАЯ ТЕМА ОЧЕНЬ ИНТЕРЕСНА, И ПОНРАВИТЬСЯ МНОГИМ!

Я надеюсь мое видео многим понравится, я старался.
Вам не трудно поставить лайк и подписаться на канал.
Всем удачи и пока!
Оставайтесь такими же крутыми!
Спасибо за просмотр:)
И ....... кстати не забудь про коммент снизу)

Рейтинг:  4.7647057
>>>
```

Рис. 36

Для примера был выбран один из видеороликов, имеющихся на просторах «Youtube». Как можно заметить, программа выдает полное описание видеоролика, и производит его загрузку в папку, в которой находится сама программа, что можно также изменить. Данная программа является хорошим примером того, что на одну кнопку можно назначить ряд функций, которые будут работать синхронно. Например: загрузка видеоролика, выбор качества изображения, и воспроизведение звукового сигнала.

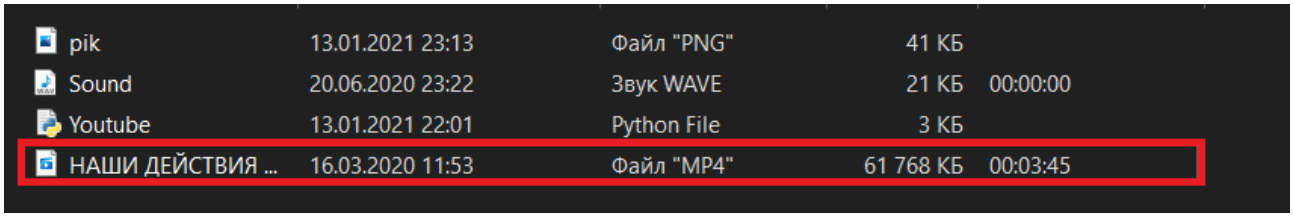

Рис. 37

ЗАКЛЮЧЕНИЕ

Наше исследование можно справедливо считать актуальным для начинающих разработчиков, а также для людей, желающих изучать «Python», углубляясь в работу с графическим интерфейсом. Потому как данная работа содержит полезные сведения о ряде проблем, которыми могут столкнуться разработчики любого уровня.

Список использованной литературы

1. Фрейд. Д. мл // Викиучебник – 2012;
2. Бентли. Дж // Жемчужины программирования (2-е издание - 2002);
3. Бёрд. Р // Жемчужины проектирования – 2017;
4. Кнут. Д // Искусство программирования – 2015;
5. Макконнелл. С // Совершенный код – 2017;