Графический интерфейс для программы на Python можно создать с помощью одной из специальных GUI библиотек:

* Tkinter
* wxPython
* PyQt
* PySide
* Kivy
* PyGTK

У каждой из этих библиотек есть свои преимущества и недостатки. Только одна из них, Tkinter, входит в стандартную поставку Python. Виджеты Tkinter не отличаются сногсшибательной стильностью – это ее единственный очевидный минус. Преимуществ у Tkinter гораздо больше, чем недостатков. Эта библиотека:

* Максимально проста в изучении и использовании.
* Имеет детальную и доступную документацию.
* Помимо базовых элементов интерфейса, содержит два мощных виджета – **Text** (многострочное текстовое поле с поддержкой форматирования) и **Canvas** («холст», на котором можно рисовать и отображать любые графические объекты).
* Включает в себя модуль **ttk**, который предоставляет в распоряжение разработчика набор дополнительных виджетов – **Combobox**, **Notebook**, **Treeview**, **Progressbar**, **Scale** и другие.
* Позволяет сделать интерфейс адаптивным.
* Отлично подходит для начинающих – поэтому на ней мы и сосредоточимся.

Создание окна приложения с Tkinter

Для создания простого окна приложения в Tkinter выполните следующие действия:

* Импортируйте модуль import tkinter.
* Создайте новый объект Tk, который представляет собой главное окно приложения.
* (Опционально) Задайте заголовок окна с помощью метода title() объекта Tk.
* (Опционально) Установите размер окна с помощью метода geometry() объекта Tk. Метод geometry() принимает строковый параметр в формате "ширина**x**высота".
* Вызовите метод mainloop() объекта Tk, чтобы запустить основной цикл GUI приложения.

Вот пример кода, который создает простое окно размером 250 на 250 пикселей с заголовком «Мое приложение»:

import tkinter as tk

# Создайте новый объект Tk

root = tk.Tk()

# Задайте заголовок окна

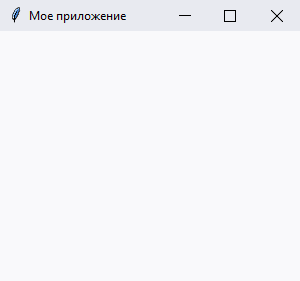
root.title("Мое приложение")

# Установите размер окна

root.geometry("250x250")

# Запустите основной цикл

root.mainloop()



Позиционирование окна в центре экрана

Чтобы разместить окно Tkinter приложения в центре экрана, необходимо:

* Воспользоваться методами **winfo\_screenwidth()** и **winfo\_screenheight()** для получения ширины и высоты экрана соответственно.
* Передать в метод **geometry()** координаты **x** и **y**, равные половине ширины и высоты экрана за минусом половины ширины и высоты окна приложения.

Код будет выглядеть так:

import tkinter as tk

# Создаем окно

root = tk.Tk()

# Получаем ширину и высоту экрана

screen\_width = root.winfo\_screenwidth()

screen\_height = root.winfo\_screenheight()

# Вычисляем координаты окна приложения

window\_width = 500

window\_height = 300

x = (screen\_width // 2) - (window\_width // 2)

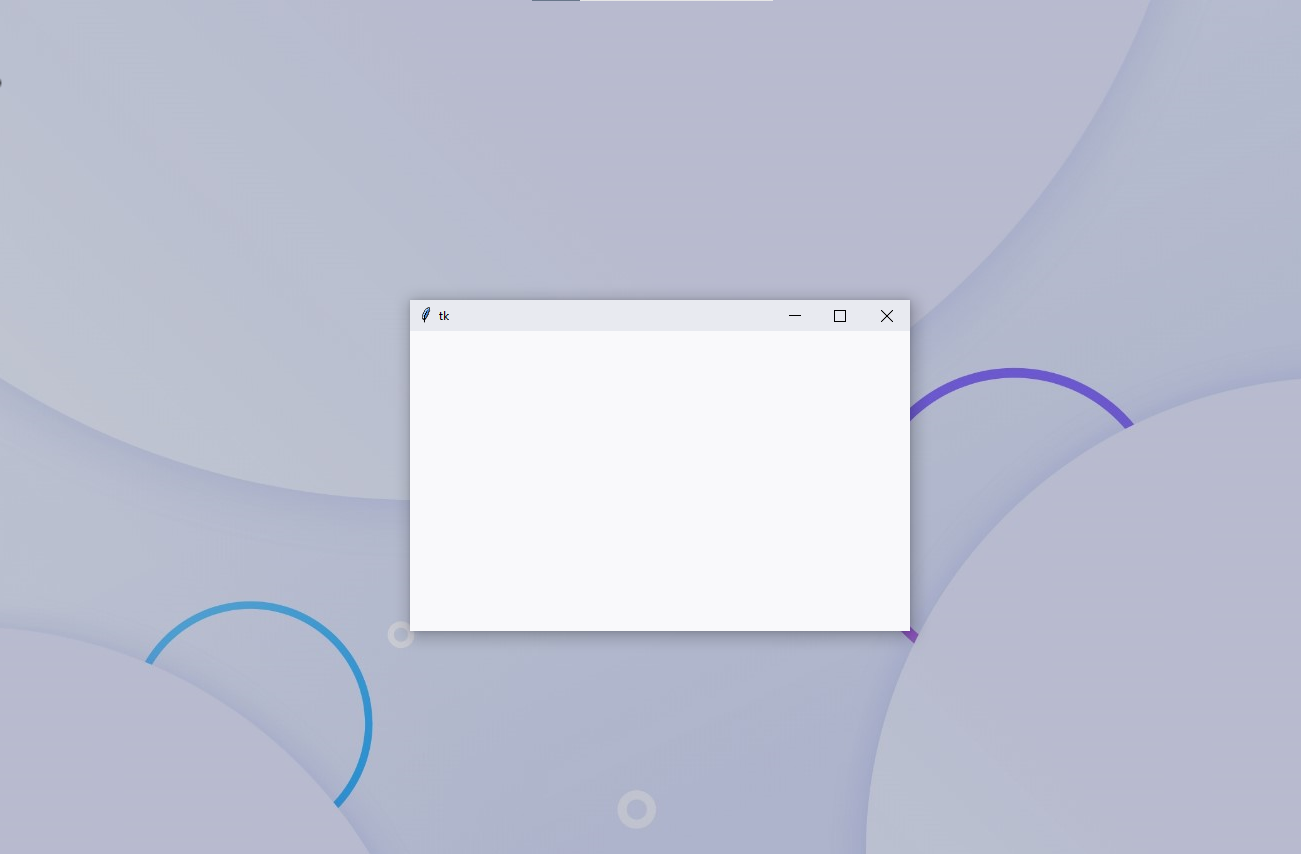
y = (screen\_height // 2) - (window\_height // 2)

root.geometry(f"{window\_width}x{window\_height}+{x}+{y}")

# Запускаем программу

root.mainloop()

Теперь окно расположено точно в центре экрана:



Размещение элементов интерфейса в Tkinter

Элементы интерфейса в Tkinter называются виджетами. Существует три основных способа расположения виджетов на поверхности окна: **pack()**, **place()** и **grid()**. Каждый из этих методов имеет свои преимущества и недостатки, и выбор оптимального способа зависит от конкретной ситуации.

**pack()** – упорядочивает виджеты по горизонтали или вертикали. Он прост в использовании, не требует дополнительных параметров (указания отступов, конкретной позиции). Подходит для создания простых интерфейсов. Недостаток – с помощью pack() проблематично реализовать сложную компоновку, например, сетку или перекрывание виджетов: для этого нужно комбинировать метод с place().

import tkinter as tk

root = tk.Tk()

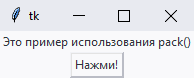
label1 = tk.Label(root, text="Это пример использования pack()")

label1.pack()

button1 = tk.Button(root, text="Нажми!")

button1.pack()

root.mainloop()



**place()**– позволяет задать точное положение и размер каждого виджета в окне. Его используют, когда необходимо точно расположить виджеты или создать перекрывающиеся элементы интерфейса. Недостаток – при изменении размеров окна или содержимого виджетов трудно сохранить расположение элементов.

import tkinter as tk

root = tk.Tk()

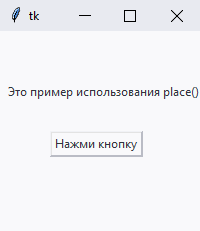
label1 = tk.Label(root, text="Это пример использования place()")

label1.place(x=5, y=50)

button1 = tk.Button(root, text="Нажми кнопку")

button1.place(x=50, y=100)

root.mainloop()



**grid()** – упорядочивает виджеты в сетку из рядов и столбцов. Самый гибкий и мощный, позволяет создавать сложные интерфейсы, состоящие из виджетов разных размеров. Сложнее в использовании, чем pack(), поскольку требует больше кода для компоновки виджетов.

import tkinter as tk

root = tk.Tk()

root.title("Пример grid()")

root.geometry("250x250")

frame = tk.Frame(root)

frame.pack(expand=True)

label1 = tk.Label(frame, text="Имя:")

label1.grid(row=0, column=0)

entry1 = tk.Entry(frame)

entry1.grid(row=0, column=1)

label2 = tk.Label(frame, text="Email:")

label2.grid(row=1, column=0)

entry2 = tk.Entry(frame)

entry2.grid(row=1, column=1)

button1 = tk.Button(frame, text="Отправить")

button1.grid(row=2, column=1)

root.update\_idletasks()

width = root.winfo\_width()

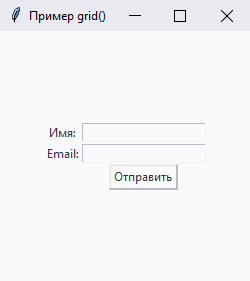
height = root.winfo\_height()

x = (root.winfo\_screenwidth() // 2) - (width // 2)

y = (root.winfo\_screenheight() // 2) - (height // 2)

root.geometry('{}x{}+{}+{}'.format(width, height, x, y))

root.mainloop()



В целом, pack() хорошо подходит для простых интерфейсов, place() – для более сложных, а grid() используют для создания сложных интерфейсов, которым нужна адаптивность, особое позиционирование или растягивание виджетов на несколько строк / столбцов.

Связывание виджетов с функциями

Чтобы при нажатии кнопки выполнялось какое-то действие, нужно связать кнопку с определенной функцией. Чаще всего для этого используются методы **command()**и **bind()**, но при необходимости к виджетам Tkinter можно привязывать выполнение анонимных и частично примененных функций. Проиллюстрируем примерами.

Метод **command()** используется для прямого связывания функции с нажатием кнопки:

import tkinter as tk

root = tk.Tk()

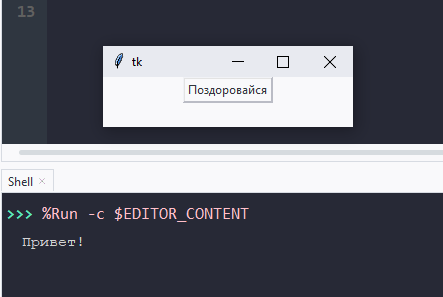
def say\_hello():

print("Привет!")

button1 = tk.Button(root, text="Поздоровайся", command=say\_hello)

button1.pack()

root.mainloop()



Метод **bind()**, по сравнению с command(), отличается большей гибкостью: его можно использовать для связывания функции с любым событием, происходящим в виджете – с нажатием кнопки, движением мыши, нажатием клавиши, изменением размера окна и так далее:

import tkinter as tk

root = tk.Tk()

def say\_hello(event=None):

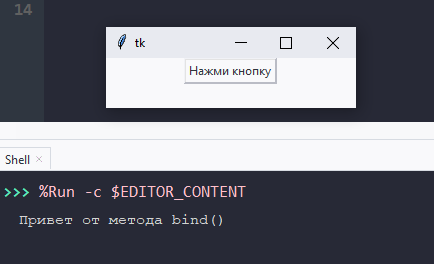
print("Привет от метода bind()")

button1 = tk.Button(root, text="Нажми кнопку")

button1.bind("<Button-1>", say\_hello)

button1.pack()

root.mainloop()



Назначение кнопке анонимной функции:

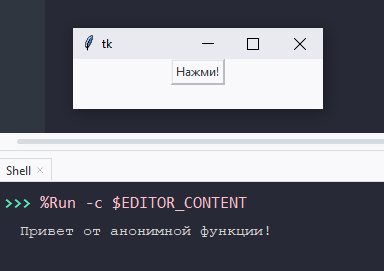
import tkinter as tk

root = tk.Tk()

button1 = tk.Button(root, text="Нажми!", command=lambda: print("Привет от анонимной функции!"))

button1.pack()

root.mainloop()



Связывание кнопки с частично примененной функцией:

import tkinter as tk

from functools import partial

root = tk.Tk()

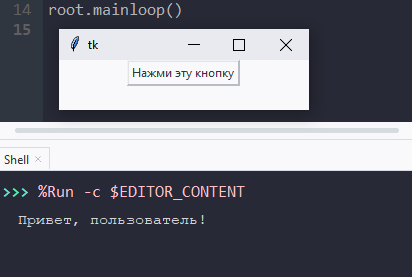
def say\_hello(name):

print(f"Привет, {name}!")

button1 = tk.Button(root, text="Нажми эту кнопку", command=partial(say\_hello, "пользователь"))

button1.pack()

root.mainloop()

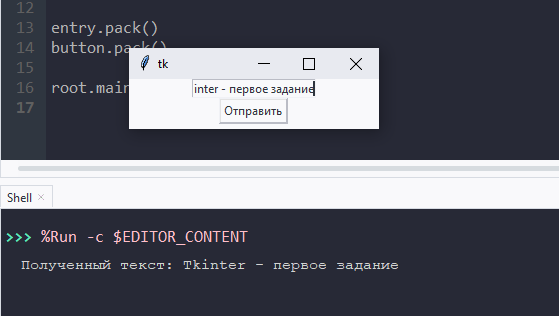


Практика

Задание 1

Создайте Tkinter интерфейс для программы, которая получает от пользователя текст с помощью виджетов **Entry**и **Button**, а затем выводит полученную строку в терминале.

**Ожидаемый результат:**



**Решение:**

import tkinter as tk

root = tk.Tk()

def button\_click():

input\_text = entry.get()

print(f"Полученный текст: {input\_text}")

entry = tk.Entry(root)

button = tk.Button(root, text="Отправить", command=button\_click)

entry.pack()

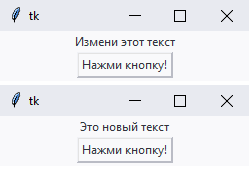
button.pack()

root.mainloop()

Задание 2

Создайте интерфейс для программы, которая изменяет текст виджета**Label**после нажатия на кнопку.

**Ожидаемый результат:**



**Решение:**

import tkinter as tk

root = tk.Tk()

label = tk.Label(root, text="Измени этот текст")

label.pack()

def change\_text():

label.config(text="Это новый текст")

button = tk.Button(root, text="Нажми кнопку!", command=change\_text)

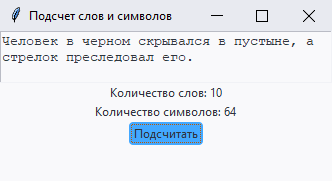
button.pack()

root.mainloop()

Задание 3

Создайте интерфейс для программы, которая получает от пользователя многострочный текст в виджете **Text** и выводит в виджетах **Label**количество слов и символов.

**Ожидаемый результат:**



import tkinter as tk

import tkinter.ttk as ttk

root = tk.Tk()

root.geometry("250x150")

root.title("Подсчет слов и символов")

def count\_words\_characters():

sentence = sentence\_entry.get("1.0", "end-1c")

words = len(sentence.split())

characters = len(sentence)

words\_label.config(text=f"Количество слов: {words}")

characters\_label.config(text=f"Количество символов: {characters}")

sentence\_entry = tk.Text(root, height=3, wrap="word")

words\_label = ttk.Label(root)

characters\_label = ttk.Label(root)

count\_button = ttk.Button(root, text="Подсчитать", command=count\_words\_characters)

sentence\_entry.pack()

words\_label.pack()

characters\_label.pack()

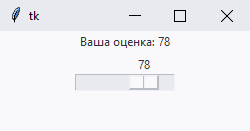
count\_button.pack()

root.mainloop()

Задание 4

Напишите программу для оценки сервиса по шкале от 0 до 100. Используйте виджет **Scale** (ttk модуль).

**Ожидаемый результат:**



**Решение:**

import tkinter as tk

root = tk.Tk()

label = tk.Label(root, text=f"Ваша оценка от 0 до 100")

label.pack()

def update\_label(value):

label.config(text=f"Ваша оценка: {value}")

scale = tk.Scale(root, from\_=0, to=100, orient="horizontal", command=update\_label)

scale.pack()

root.mainloop()

Задание 5

Создайте индикатор выполнения задачи с помощью виджета **Progressbar** из модуля ttk. Индикатор должен обнуляться спустя 5 секунд.

**Ожидаемый результат:**

🐍 Самоучитель по Python для начинающих. Часть 20: Графический интерфейс на Tkinter

**Решение:**

import tkinter as tk

from tkinter import ttk

root = tk.Tk()

progressbar = ttk.Progressbar(root, orient="horizontal", length=200, mode="determinate")

progressbar.start()

def stop\_progressbar():

if root.winfo\_exists() and progressbar.winfo\_exists():

progressbar.stop()

progressbar["value"] = 0

root.quit()

root.destroy()

if root.winfo\_exists() and progressbar.winfo\_exists():

root.after(5000, stop\_progressbar)

progressbar.pack()

def exit\_app():

stop\_progressbar()

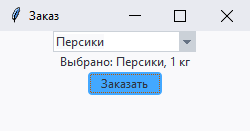
root.protocol("WM\_DELETE\_WINDOW", exit\_app)

root.mainloop()

Задание 6

Создайте Tkinter интерфейс для программы заказа фруктов. Для вывода списка фруктов используйте виджет **Combobox** из модуля ttk.

**Ожидаемый результат:**



**Решение:**

import tkinter as tk

import tkinter.ttk as ttk

root = tk.Tk()

root.title("Заказ")

def select\_item():

selected\_item = combobox.get()

selection\_label.config(text=f"Выбрано: {selected\_item}, 1 кг")

items = ["Яблоки", "Апельсины", "Виноград", "Персики", "Клубника"]

combobox = ttk.Combobox(root, values=items)

selection\_label = ttk.Label(root)

select\_button = ttk.Button(root, text="Заказать", command=select\_item)

combobox.pack()

selection\_label.pack()

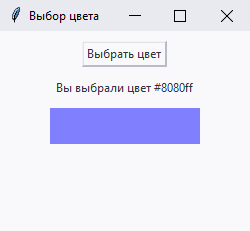
select\_button.pack()

root.mainloop()

Задание 7

Напишите GUI программу, которая позволяет пользователю выбрать цвет из палитры, и отображает его HEX-значение.

**Ожидаемый результат:**



**Решение:**

import tkinter as tk

import tkinter.colorchooser as cc

root = tk.Tk()

root.geometry("250x200")

root.title("Выбор цвета")

def choose\_color():

color = cc.askcolor()[1]

color\_label.config(text=f"Вы выбрали цвет {color}")

second\_label.config(bg=color)

color\_button = tk.Button(root, text="Выбрать цвет", command=choose\_color)

color\_label = tk.Label(root, text="Нажмите кнопку, чтобы выбрать цвет")

second\_label = tk.Label(root, text="\t\t\t\n\t\t\t")

color\_button.pack(pady=10)

color\_label.pack()

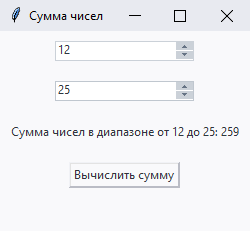
second\_label.pack(pady=10)

root.mainloop()

Задание 8

Напишите программу, которая использует виджеты **Spinbox** для получения двух чисел **a** и **b** (0 <= **a**<= 100, 0 <= **b**<= 100), а затем выводит сумму всех целых чисел в диапазоне от a до b **включительно**.

**Ожидаемый результат:**



**Решение:**

import tkinter as tk

import tkinter.ttk as ttk

class SumApp:

def \_\_init\_\_(self, root):

self.root = root

root.geometry("250x200")

self.root.title("Сумма чисел")

self.start\_spinbox = ttk.Spinbox(root, from\_=0, to=100, increment=1)

self.end\_spinbox = ttk.Spinbox(root, from\_=0, to=100, increment=1)

self.sum\_label = tk.Label(root, text="")

self.calc\_button = tk.Button(root, text="Вычислить сумму", command=self.calculate\_sum)

self.start\_spinbox.pack(pady=10)

self.end\_spinbox.pack(pady=10)

self.sum\_label.pack(pady=10)

self.calc\_button.pack(pady=10)

def calculate\_sum(self):

start = int(self.start\_spinbox.get())

end = int(self.end\_spinbox.get())

sum\_ = sum(range(start, end+1))

self.sum\_label.configure(text=f"Сумма чисел в диапазоне от {start} до {end}: {sum\_}")

root = tk.Tk()

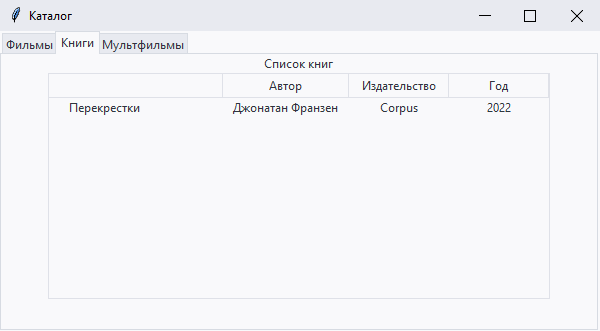
app = SumApp(root)

root.mainloop()

Задание 9

Создайте GUI интерфейс для программы-каталога фильмов, книг и мультфильмов. Используйте виджеты **Notebook** и **Treeview** для вывода объектов разного типа на отдельных вкладках. Реализуйте всплывающее окно для вывода полной информации об издании.

**Ожидаемый результат:**



**Решение:**

import tkinter as tk

from tkinter import ttk

def retrieve\_details(event):

item = event.widget.focus()

values = event.widget.item(item)['values']

details = f"Информация об издании:\n\n"

for i in range(len(values)):

details += f"{event.widget['columns'][i].capitalize()}: {values[i]}\n"

popup = tk.Toplevel(root)

popup.title(event.widget.item(item)['text'])

popup.geometry("250x150")

popup\_label = ttk.Label(popup, text=details)

popup\_label.pack(pady=10)

close\_button = ttk.Button(popup, text="Закрыть", command=popup.destroy)

close\_button.pack()

root = tk.Tk()

root.title("Каталог")

root.geometry("600x300")

notebook = ttk.Notebook(root)

movies\_tab = ttk.Frame(notebook)

books\_tab = ttk.Frame(notebook)

animation\_tab = ttk.Frame(notebook)

notebook.add(movies\_tab, text="Фильмы")

notebook.add(books\_tab, text="Книги")

notebook.add(animation\_tab, text="Мультфильмы")

movies\_desc = ttk.Label(movies\_tab, text="Список фильмов")

books\_desc = ttk.Label(books\_tab, text="Список книг")

animation\_desc = ttk.Label(animation\_tab, text="Список мультфильмов")

movies\_list = ttk.Treeview(movies\_tab)

books\_list = ttk.Treeview(books\_tab)

animation\_list = ttk.Treeview(animation\_tab)

movies\_list['columns'] = ('Жанр', 'Режиссер', 'Год')

books\_list['columns'] = ('Автор', 'Издательство', 'Год')

animation\_list['columns'] = ('Жанр', 'Студия', 'Рейтинг')

for tree in [movies\_list, books\_list, animation\_list]:

for i, col in enumerate(tree['columns']):

tree.column(col, width=100, anchor='center')

tree.heading(col, text=col.capitalize())

tree.bind("<Double-1>", retrieve\_details)

movies\_list.insert('', 'end', text='М3ган', values=('Фантастика', 'Джерард Джонстоун', '2022'))

books\_list.insert('', 'end', text='Перекрестки', values=('Джонатан Франзен', 'Corpus', '2022'))

animation\_list.insert('', 'end', text='Пиноккио Гильермо Дель Торо', values=('фэнтези', 'Disney+', '8.2'))

movies\_desc.pack()

movies\_list.pack()

books\_desc.pack()

books\_list.pack()

animation\_desc.pack()

animation\_list.pack()

notebook.pack(fill='both', expand=True)

root.mainloop()

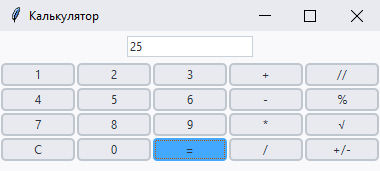
Задание 10

Напишите GUI калькулятор, который:

* Выполняет основные арифметические операции – сложение, вычитание, деление и умножение.
* Поддерживает операции с отрицательными числами, извлечение квадратного корня, деление с остатком и целочисленное деление.

Кроме того, программа должна поддерживать очистку ввода.

**Ожидаемый результат:**



**Решение:**

import tkinter as tk

from tkinter import ttk

from math import sqrt

class Calculator:

def \_\_init\_\_(self, master):

self.master = master

self.master.title("Калькулятор")

self.master.geometry("380x140")

self.number\_entry = ttk.Entry(self.master, width=20)

self.number\_entry.grid(row=0, column=0, columnspan=5, padx=5, pady=5)

self.button\_1 = ttk.Button(self.master, text="1", command=lambda: self.button\_click(1))

self.button\_2 = ttk.Button(self.master, text="2", command=lambda: self.