|  |  |  |
| --- | --- | --- |
|  |  | УТВЕРЖДАЮ |
|  |  | Преподаватель |
|  |  | \_\_\_\_\_\_\_\_\_\_\_ Дергунов Е.А. |
|  |  | “\_\_\_\_\_” \_\_\_\_\_\_\_\_\_\_\_\_20\_\_ |

ПРОГРАММА «РИСОВАНИЕ ГРАФИКА ФУНКЦИИ Y=AX^3»

ТЕКСТ ПРОГРАММЫ

ЛИСТ УТВЕРЖДЕНИЯА.В.00001-01 ТЗ 01

R.G.0001-01 02 24 ЛУ

Инв. № подл.

Подпись и дата

Взам. инв. №

Инв. № дубл.

Подпись и дата

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  | Разработчик |
|  |  |  |
|  |  | \_\_\_\_\_\_\_\_\_\_\_\_Володин И.В. |
|  |  | “\_\_\_\_\_” \_\_\_\_\_\_\_\_\_\_\_\_20\_\_ |
|  |  |  |
|  |  | Тестировщик |
|  |  |  |
|  |  | \_\_\_\_\_\_\_\_\_\_\_\_Володин И.В. |
|  |  | “\_\_\_\_\_” \_\_\_\_\_\_\_\_\_\_\_\_20\_\_ |
|  |  |  |
|  |  | Ответственный за эксплуатацию |
|  |  |  |
|  |  | \_\_\_\_\_\_\_\_\_\_\_\_Володин И.В. |
|  |  | “\_\_\_\_\_” \_\_\_\_\_\_\_\_\_\_\_\_20\_\_ |

УТВЕРЖДЕНО

R.G.0001-01 02 24

ПРОГРАММА РИСОВАНИЕ ГРАФИКА ФУНКЦИИ Y=AX^3

ТЕКСТ ПРОГРАММЫ

R.G.0001-01 02 24

**Инв. № подл.**

**Подпись и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подпись и дата**

АННОТАЦИЯ

В данном программном документе приведено техническое задание на разработку программы «Рисование графика функции y = ax^3». Текст программы реализован в виде символической записи на исходном языке. Исходным языком данной разработки является Python. Среда разработки, компилятор – PyCharm Community Edition.

Основной функцией программы является рисование графика функции у = ах^3. Программа предоставляет пользователю возможность вводить значение «а» и получать нарисованный график с использованием простого и интуитивно понятного интерфейса.

Основная задача программы состоит в обеспечении удобного и надежного инструмента для на рисовки графика функции у = ах^3. Программа позволяет пользователям быстро и точно рисовать графики функции у = ах^3.

В разделе «Текст программы на исходном языке» указана символическая запись на исходном языке.

Оформление программного документа «Текст программы» произведено по требованиям ЕСПД (ГОСТ 19.101-77, ГОСТ 19.103-77, ГОСТ 19.104-78, ГОСТ 19.105-78, ГОСТ 19.106-78, ГОСТ 19.401-78, ГОСТ 19.604-78).

Содержание

|  |  |
| --- | --- |
| 1 Текст программы на исходном языке | 3 |
| Лист регистрации изменений | 14 |

1 ТЕКСТ ПРОГРАММЫ НА ИСХОДНОМ ЯЗЫКЕ

Импорт необходимых модулей и классов из стандартной библиотеки и других модулей.

----------------------------------------------------------------------------------------------------------

import tkinter as tk

import numpy as np

import tkinter.messagebox as messagebox

from PIL import Image

import matplotlib.pyplot as plt

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

from tkinter import filedialog

import speech\_recognition as sr

from gtts import gTTS

import pygame

import os

import sys

----------------------------------------------------------------------------------------------------------

Ввод функции speak(text), которая создает объект gTTS с переданным текстом и указанием языка 'ru'. Создает файл output.mp3, в котором сохраняет речь, используя метод save() объекта gTTS. Инициализирует модуль pygame.mixer для управления звуковым воспроизведением. Загружает файл output.mp3 в pygame.mixer.music с помощью метода load(). Воспроизводит звуковой файл с помощью метода play() объекта pygame.mixer.music. Запускает бесконечный цикл while для ожидания завершения воспроизведения звука. Завершает воспроизведение с помощью метода stop() объекта pygame.mixer.music. Выполняет очистку и выгрузку модуля pygame.mixer. Удаляет временный файл output.mp3 с помощью os.remove().

----------------------------------------------------------------------------------------------------------

def speak(text):

tts = gTTS(text=text, lang='ru')

tts.save('output.mp3')

pygame.mixer.init()

pygame.mixer.music.load('output.mp3')

pygame.mixer.music.play()

while pygame.mixer.music.get\_busy():

continue

pygame.mixer.music.stop()

pygame.mixer.quit()

os.remove('output.mp3')

----------------------------------------------------------------------------------------------------------

Ввод переменных с желаемый текстом для произнесения программой при запуске и включении программы.

----------------------------------------------------------------------------------------------------------

text\_to\_speak = "Запуск программы"

text\_to\_speak1 = "Выключение программы"

----------------------------------------------------------------------------------------------------------

- Создание класса MainWindow, который наследуется от класса tk.Tk и представляет главное окно приложения. Конструктор класса \_\_init\_\_(self):

- вызывает конструктор суперкласса tk.Tk с помощью super().\_\_init\_\_() для инициализации окна;

- устанавливает заголовок окна на "График функции y = ax^3" с помощью метода title();

- устанавливает размеры окна на 663 пикселя по ширине и 610 пикселей по высоте с помощью метода geometry();

- устанавливает иконку приложения;

- устанавливает шрифт для всех виджетов приложения;

- устанавливает цвет фона окна на «gainsboro».

Далее, в конструкторе:

- создается экземпляр Figure и Axes из модуля matplotlib.pyplot для построения графиков.

- создается экземпляр FigureCanvasTkAgg для встраивания графического холста Figure в окно приложения;

- создается меню приложения с помощью класса Menu;

- создаются подменю «Файл» и «Справка»;

- устанавливается созданное меню в главном окне с помощью метода config(menu=self.menu\_bar);

- создается список self.plot\_data для хранения данных графиков;

- создается метка self.label\_a с текстом «Значение а=» и фоном «gainsboro»;

- создается текстовое поле self.edit\_a для ввода значения «а»;

- создается кнопка self.button\_plot с текстом «Нарисовать график» и цветом фона «orange»»

- размещаются метка, текстовое поле и кнопка на главном окне с помощью методов place() и relx, rely, anchor для задания относительных координат и выравнивания;

- вызывается метод center\_window() для восстановления позиции окна и размещения его по центру экрана;

----------------------------------------------------------------------------------------------------------

class MainWindow(tk.Tk):

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

super().\_\_init\_\_()

self.title("График функции y = ax^3")

self.geometry("663x610")

# Установка иконки приложения

self.iconbitmap("2.ico")

# Установка шрифта

self.option\_add("\*Font", "Calibri 12")

# Установка цвета фона

self.configure(bg="gainsboro")

# Создание экземпляра Figure и Axes

self.figure, self.axes = plt.subplots()

self.figure\_canvas = FigureCanvasTkAgg(self.figure, master=self)

self.figure\_canvas.get\_tk\_widget().place(relx=0.5, rely=0.57, anchor=tk.CENTER)

# Создание меню

self.menu\_bar = tk.Menu(self)

# Меню "Файл"

self.file\_menu = tk.Menu(self.menu\_bar, tearoff=0)

self.file\_menu.add\_command(label="Сохранить графики", command=self.save\_graphs)

self.file\_menu.add\_command(label="Выход", command=self.quit)

self.menu\_bar.add\_cascade(label="Файл", menu=self.file\_menu)

# Меню "Справка"

self.help\_menu = tk.Menu(self.menu\_bar, tearoff=0)

self.help\_menu.add\_command(label="О программе", command=self.show\_about\_dialog)

self.menu\_bar.add\_cascade(label="Справка", menu=self.help\_menu)

# Установка созданного меню

self.config(menu=self.menu\_bar)

# Список для хранения данных графиков

self.plot\_data = []

self.label\_a = tk.Label(self, text="Значение а =", bg="gainsboro")

self.edit\_a = tk.Entry(self)

self.button\_plot = tk.Button(self, text="Нарисовать график", command=self.plot\_graph, bg="orange")

self.label\_a.place(relx=0.3, rely=0.04, anchor=tk.CENTER)

self.edit\_a.place(relx=0.5, rely=0.04, anchor=tk.CENTER)

self.button\_plot.place(relx=0.5, rely=0.11, anchor=tk.CENTER)

# Восстановление позиции окна и размещение по центру экрана

self.center\_window()

----------------------------------------------------------------------------------------------------------

Ввод функции show\_about\_dialog(self) для создания справки о программе и отображения информационного диалогового окна с помощью функции showinfo() из модуля messagebox.

----------------------------------------------------------------------------------------------------------

def show\_about\_dialog(self):

messagebox.showinfo("О программе",

"График функции y = ax^3\nВерсия: 1.0.0\nАвтор: Илья\nКомпания: ILYA Company")

----------------------------------------------------------------------------------------------------------

Ввод функции quit(self) для вызова метода save\_window\_position() для сохранения позиции окна перед его закрытием и вызова метода destroy(), который уничтожает главное окно приложения.

----------------------------------------------------------------------------------------------------------

def quit(self):

# Сохранение позиции окна перед закрытием

self.save\_window\_position()

self.destroy()

----------------------------------------------------------------------------------------------------------

Ввод функция plot\_graph(self) для получения значений переменных «a» из текстового поля self.edit\_a и преобразования его в тип float, создания одномерного массива «x» со 100 равномерно распределенными значениями от -10 до 10, вычисления массива «y» путем возведения x в куб и умножения на значение «a», добавления данные графика (x, y, f"y = {a}x^3") в список self.plot\_data, очищения предыдущего графика с помощью метода clear(), отображения всех графиков из self.plot\_data на одном графическом холсте, для установки меток осей с помощью методов set\_xlabel() и set\_ylabel() и добавления легенд к графику с помощью метода legend() и для отображения графиков на графическом холсте с помощью метода draw().

----------------------------------------------------------------------------------------------------------

def plot\_graph(self):

a = float(self.edit\_a.get())

x = np.linspace(-10, 10, 100)

y = a \* x \*\* 3

# Устанавливаем данные графика

self.plot\_data.append((x, y, f"y = {a}x^3"))

# Очищаем предыдущий график

self.axes.clear()

# Показываем все графики на одной плоскости

for i, data in enumerate(self.plot\_data):

self.axes.plot(data[0], data[1], label=data[2])

# Устанавливаем метки осей

self.axes.set\_xlabel('x')

self.axes.set\_ylabel('y')

# Добавляем легенду

self.axes.legend()

# Показовываем график

self.figure\_canvas.draw()

----------------------------------------------------------------------------------------------------------

Ввод функции center\_window(self) для получения размеров экрана с помощью функций winfo\_screenwidth() и winfo\_screenheight(), размеров окна, необходимых для отображения содержимого окна, с помощью функций winfo\_reqwidth() и winfo\_reqheight(), вычисления координат x и y, чтобы окно было размещено по центру экрана. Для установки геометрии окна с помощью метода geometry(), чтобы окно было размещено посередине экрана с заданными координатами x и y.

----------------------------------------------------------------------------------------------------------

def center\_window(self):

# Размещение окна по центру экрана

screen\_width = self.winfo\_screenwidth()

screen\_height = self.winfo\_screenheight()

window\_width = self.winfo\_reqwidth()

window\_height = self.winfo\_reqheight()

x = int((screen\_width - window\_width) / 2) - 30

y = int((screen\_height - window\_height) / 2) - 150

self.geometry(f"+{x}+{y}")

----------------------------------------------------------------------------------------------------------

Ввод функции save\_graphs(self) для проверки наличия данных графиков в переменной self.plot\_data, если данных нет, выводится предупреждение пользователю и функция завершается, если да, то открывается диалоговое окно сохранения файла, где пользователь может выбрать имя и расширение файла для сохранения графиков. Если пользователь выбрал файл для сохранения:

- создается новый экземпляр Figure и Axes с помощью subplots();

- каждый график из self.plot\_data рисуется на Axes с помощью метода plot();

- устанавливаются метки осей с помощью set\_xlabel() и set\_ylabel();

- добавляется легенда на график с помощью legend();

- график сохраняется в файле PNG с помощью savefig();

- график очищается с помощью clear();

- показывается информационное сообщение о успешном сохранении графиков.

Если возникает исключение в процессе сохранения графиков, выводится сообщение об ошибке с описанием исключения.

----------------------------------------------------------------------------------------------------------

def save\_graphs(self):

# Проверка наличия данных графиков

if not self.plot\_data:

messagebox.showwarning("Предупреждение", "Нет данных графиков для сохранения.")

return

# Выбор файла для сохранения

filename = filedialog.asksaveasfilename(defaultextension=".png",

filetypes=(("PNG", "\*.png"), ("Все файлы", "\*.\*")))

if filename:

try:

# Создание нового экземпляра Figure и Axes

figure, axes = plt.subplots()

# Сохранение каждого графика в отдельном файле

for data in self.plot\_data:

x, y, label = data

axes.plot(x, y, label=label)

# Устанавливаем метки осей

axes.set\_xlabel('x')

axes.set\_ylabel('y')

# Добавляем легенду

axes.legend()

# Сохранение графика в файле PNG

figure.savefig(filename)

# Очистка графика

axes.clear()

messagebox.showinfo("Сохранение успешно", "Графики успешно сохранены.")

except Exception as e:

messagebox.showerror("Ошибка сохранения", f"Ошибка при сохранении графиков: {str(e)}")

# Очищаем предыдущий график

self.axes.clear()

# Показываем все графики на одной плоскости

for data in self.plot\_data:

self.axes.plot(data[0], data[1], label=data[2])

# Устанавливаем метки осей

self.axes.set\_xlabel('x')

self.axes.set\_ylabel('y')

# Добавляем легенду

self.axes.legend()

# Показываем график

self.figure\_canvas.draw()

messagebox.showinfo("Открытие успешно", "Графики успешно открыты.")

except Exception as e:

messagebox.showerror("Ошибка открытия", f"Ошибка при открытии графиков: {str(e)}")

----------------------------------------------------------------------------------------------------------

Ввод функции on\_closing() для вызова функции speak(text\_to\_speak1), которая используется для синтеза и воспроизведения речи на основе текста, переданного в переменную text\_to\_speak1. Вызов метода destroy() для объекта window, который уничтожает главное окно приложения. Вызов функции exit(), которая приводит к завершению выполнения программы.

----------------------------------------------------------------------------------------------------------

def on\_closing():

speak(text\_to\_speak1)

window.destroy()

exit()

----------------------------------------------------------------------------------------------------------

Вызов главной функции main для вызова функции speak(text\_to\_speak), которая используется для синтеза и воспроизведения речи на основе текста, переданного в переменную text\_to\_speak. Создания экземпляра класса MainWindow и присваивание переменной window. Привязывание с помощью метода protocol функция on\_closing к событию закрытия окна. Запуск основного цикла обработки событий с помощью метода mainloop(), который ожидает и обрабатывает события пользовательского взаимодействия с окном, пока окно не будет закрыто.

----------------------------------------------------------------------------------------------------------

if \_\_name\_\_ == "\_\_main\_\_":

speak(text\_to\_speak)

window = MainWindow()

window.protocol("WM\_DELETE\_WINDOW", on\_closing)

window.mainloop()

----------------------------------------------------------------------------------------------------------

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Лист регистрации изменений | | | | | | | | | |
| Номера листов (страниц) | | | | | Всего  листов  (страниц)  в докум | №  документа | Входящий  № сопрово  дительного  документа  и дата | Подп. | Дата |
| Изм | изменен  ных | заме  ненных | новых | анулиро  ванных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |