

Разработка файлового менеджера с графическим интерфейсом на Python

Эрдман Александр Алексеевич

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

В данном исследовании рассматривается процесс создания файлового менеджера с графической оболочкой. Программа реализована на языке программирования Python с применением модуля Tkinter. Результатом исследования является практическое применение приложения и описание его работы.

Ключевые слова: python, tkinter, графическое приложение

Development of a file manager with a graphical interface in Python

Erdman Alexander Alekseevich

Sholom-Aleichem Priamursky State University

Student

Abstract

This study examines the process of creating a file manager with a graphical shell. The program is implemented in the Python programming language using the Tkinter module. The result of the research is the practical application of the application and a description of its work.

Keywords: python, tkinter, graphical application

1 Введение

1.1 Актуальность

В современном мире управление файлами и папками на компьютере становится всё более сложной задачей, особенно с увеличением объёма данных и разнообразия форматов. Пользователи сталкиваются с необходимостью эффективного управления своими файлами, будь то документы, изображения, видео или другие типы данных. В данном случае разработка удобных и функциональных файловых менеджеров становится актуальной задачей. Файловые менеджеры предоставляют пользователям инструменты для выполнения различных операций с файлами, таких как копирование, перемещение, удаление и создание новых файлов и папок. В данной статье рассматривается процесс создания программы файлового менеджера на базе библиотеки Tkinter для создания графического интерфейса и стандартных библиотек Python для работы с файловой системой.

1.2 Обзор исследований

А.А. Новичихина в статье рассмотрела возможность создания простых графических интерфейсов и описано создание приложения для рисования средствами tkinter в Python [1]. Е.С. Котыбаев описал способ создания приложений с графическим интерфейсом на языке программирования Python с помощью встроенной библиотеки Tkinter [2]. В.С. Чернова, Е.Б. Сидорова и Т.Г. Орешкина в совместном исследовании реализовали приложения с графическим интерфейсом на языке программирования Python 3.8 с использованием набора библиотек PyQt [3]. К.И. Чудинов в своём труде привел пример реализации приложения на языке Python для обработки данных, полученных в ходе виртуальных экспериментов в программном пакете ANSYS Fluent [4]. И.С. Андриенко в статье исследовал процесс разработки приложения для редактирования текста с использованием библиотеки Kivy на языке программирования Python [5].

1.3 Цель исследования

Целью исследования является реализация файлового менеджера и его практическое применение.

2 Материалы и методы

Для создания программы используется язык программирования Python со стандартной библиотекой Tkinter. В качестве среды разработки выступает PyCharm.

3 Результаты и обсуждения

Разработка программы файлового менеджера начинается с определения основных требований и функциональности, которую она должна предоставлять пользователям. Основные задачи включают создание графического интерфейса для отображения файлов и папок, реализацию функций сортировки, а также возможность выполнения различных операций с файлами, таких как копирование, перемещение, удаление и создание новых файлов и папок. Для достижения этих целей используется библиотека Tkinter, которая предоставляет мощные инструменты для создания графических интерфейсов в Python.

Первым шагом в разработке программы является создание основного класса “FileManager”, который отвечает за инициализацию графического интерфейса и управление файловой системой (рис. 1). В конструкторе класса “__init__” инициализируются основные параметры, такие как текущий путь и критерии сортировки, а также вызываются методы для создания виджетов и обновления списка файлов.

```
class FileManager:
    def __init__(self, root):
        self.root = root
        self.root.title("Файловый менеджер")
        self.root.geometry("800x600")

        self.current_path = os.getcwd()
        self.sort_criteria = tk.StringVar(value="Имя")

        self.create_widgets()
        self.update_file_list()
```

Рисунок 1. Класс “FileManager”

Следующим шагом является реализация функции “create_widgets” (рис. 2).

```
def create_widgets(self):
    self.path_label = tk.Label(self.root, text=self.current_path)
    self.path_label.pack(fill=tk.X, padx=10, pady=10)
    self.sort_frame = tk.Frame(self.root)
    self.sort_frame.pack(fill=tk.X, padx=10, pady=5)
    self.sort_label = tk.Label(self.sort_frame, text="Сортировать по:")
    self.sort_label.pack(side=tk.LEFT, padx=5)
    self.sort_menu = tk.OptionMenu(self.sort_frame, self.sort_criteria, values="Имя", values="Тип", values="Дата создания",
    self.sort_menu.pack(side=tk.LEFT, padx=5)

    self.tree = ttk.Treeview(self.root, columns=("Имя", "Тип", "Дата создания"), show='headings')
    self.tree.heading("Имя", text="Имя")
    self.tree.heading("Тип", text="Тип")
    self.tree.heading("Дата создания", text="Дата создания")
    self.tree.column("Имя", width=300)
    self.tree.column("Тип", width=100)
    self.tree.column("Дата создания", width=200)
    self.tree.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)
    self.tree.bind('<<TreeviewSelect>>', self.on_select)

    self.button_frame = tk.Frame(self.root)
    self.button_frame.pack(fill=tk.X, padx=10, pady=10)
    self.back_button = tk.Button(self.button_frame, text="Назад", command=self.go_back)
    self.back_button.pack(side=tk.LEFT, padx=5)
    self.open_button = tk.Button(self.button_frame, text="Открыть", command=self.open_file)
    self.open_button.pack(side=tk.LEFT, padx=5)
    self.delete_button = tk.Button(self.button_frame, text="Удалить", command=self.delete_file)
    self.delete_button.pack(side=tk.LEFT, padx=5)
    self.new_file_button = tk.Button(self.button_frame, text="Новый файл", command=self.create_new_file)
    self.new_file_button.pack(side=tk.LEFT, padx=5)
    self.new_folder_button = tk.Button(self.button_frame, text="Новая папка", command=self.create_new_folder)
    self.new_folder_button.pack(side=tk.LEFT, padx=5)
    self.copy_button = tk.Button(self.button_frame, text="Копировать", command=self.copy_file)
    self.copy_button.pack(side=tk.LEFT, padx=5)
    self.move_button = tk.Button(self.button_frame, text="Переместить", command=self.move_file)
    self.move_button.pack(side=tk.LEFT, padx=5)
```

Рисунок 2. Метод “create_widgets”

Метод “create_widgets” является центральным элементом в создании графического интерфейса программы. Он отвечает за инициализацию и размещение всех виджетов, которые будут отображаться на главном окне приложения. Здесь создаются метки, кнопки, меню сортировки и дерево для отображения файлов и папок. Каждый виджет размещается в соответствующем фрейме и настраивается для обеспечения удобного интерфейса.

Метка текущего пути “path_label” отображает текущий путь в файловой системе, позволяя пользователю всегда видеть, в какой директории он находится. Область для меню сортировки “sort_frame” содержит элементы для сортировки файлов и папок, включая метку с текстом “Сортировать по:” и выпадающее меню, которое позволяет пользователю выбрать критерий сортировки: по имени, типу или дате создания. При изменении критерия вызывается метод “update_file_list”, который обновляет список файлов и папок в соответствии с выбранным критерием.

Дерево для отображения файлов и папок “tree” имеет три колонки: “Имя”, “Тип” и “Дата создания”. Каждая колонка настраивается с заголовком и шириной. Дерево размещается в центральной части окна и заполняет всё доступное пространство. При выборе элемента в дереве вызывается метод “on_select”, который сохраняет имя выбранного элемента для последующих операций. Фрейм для кнопок “button_frame” содержит кнопки для выполнения различных операций с файлами и папками, таких как переход назад, открытие файла, удаление, создание нового файла, создание новой папки, копирование и перемещение файла. Каждая кнопка размещается внутри области кнопок слева и имеет отступ. При нажатии на кнопку вызывается соответствующий метод, который выполняет нужную операцию.

Следующим этапом разработки является реализация функции “update_file_list”, которая отвечает за обновление списка файлов и папок в текущей директории (рис. 3).

```
def update_file_list(self, *args):
    for i in self.tree.get_children():
        self.tree.delete(i)

    items = os.listdir(self.current_path)

    if self.sort_criteria.get() == "Имя":
        items.sort()
    elif self.sort_criteria.get() == "Тип":
        items.sort(key=lambda x: 'dir' if os.path.isdir(os.path.join(self.current_path, x)) else 'file')
    elif self.sort_criteria.get() == "Дата создания":
        items.sort(key=lambda x: os.path.getctime(os.path.join(self.current_path, x)))

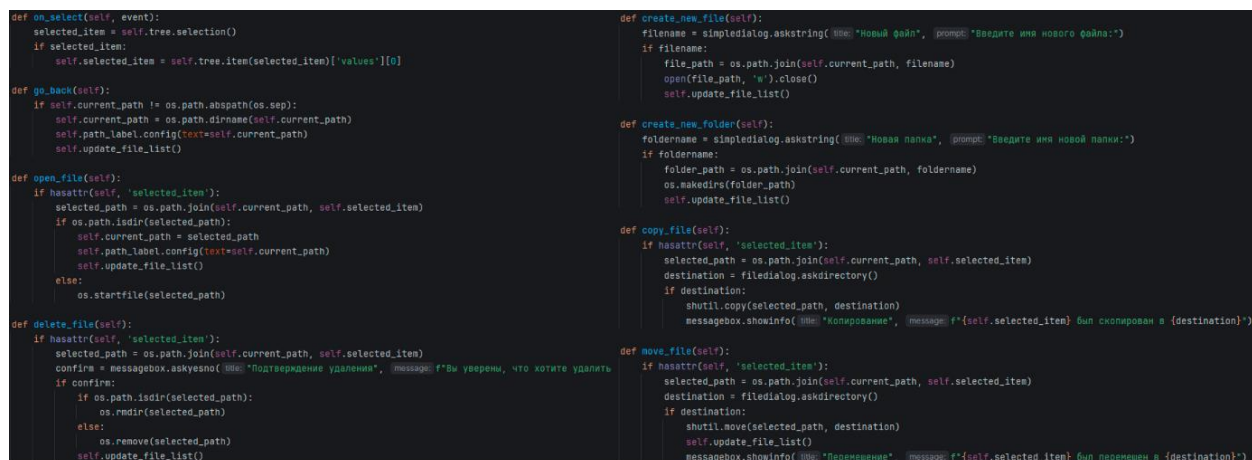
    for item in items:
        item_path = os.path.join(self.current_path, item)
        item_type = 'Папка' if os.path.isdir(item_path) else 'Файл'
        creation_time = datetime.fromtimestamp(os.path.getctime(item_path)).strftime('%Y-%m-%d %H:%M:%S')
        self.tree.insert(parent="", index="end", values=(item, item_type, creation_time))
```

Рисунок 3. Метод обновления списка файлов

Данный метод очищает текущее содержимое дерева, получает список файлов и папок, сортирует их в соответствии с выбранным критерием и добавляет в дерево.

Следующие функции “on_select”, “go_back”, “open_file”, “delete_file”, “create_new_file”, “create_new_folder”, “copy_file” и “move_file” являются ключевыми компонентами функциональности файлового менеджера (рис. 4).

Они обеспечивают основные операции с файлами и папками, такие как выбор элемента, навигация по директориям, открытие файлов, удаление, создание новых файлов и папок, а также копирование и перемещение файлов.



```
def on_select(self, event):
    selected_item = self.tree.selection()
    if selected_item:
        self.selected_item = self.tree.item(selected_item)['values'][0]

def go_back(self):
    if self.current_path != os.path.abspath(os.sep):
        self.current_path = os.path.dirname(self.current_path)
        self.path_label.config(text=self.current_path)
        self.update_file_list()

def open_file(self):
    if hasattr(self, 'selected_item'):
        selected_path = os.path.join(self.current_path, self.selected_item)
        if os.path.isdir(selected_path):
            self.current_path = selected_path
            self.path_label.config(text=self.current_path)
            self.update_file_list()
        else:
            os.startfile(selected_path)

def delete_file(self):
    if hasattr(self, 'selected_item'):
        selected_path = os.path.join(self.current_path, self.selected_item)
        confirm = messagebox.askyesno("Подтверждение удаления", "Вы уверены, что хотите удалить")
        if confirm:
            if os.path.isdir(selected_path):
                os.rmdir(selected_path)
            else:
                os.remove(selected_path)
            self.update_file_list()

def create_new_file(self):
    filename = simpledialog.askstring("Новый файл", "Введите имя нового файла:")
    if filename:
        file_path = os.path.join(self.current_path, filename)
        open(file_path, 'w').close()
        self.update_file_list()

def create_new_folder(self):
    foldername = simpledialog.askstring("Новая папка", "Введите имя новой папки:")
    if foldername:
        folder_path = os.path.join(self.current_path, foldername)
        os.makedirs(folder_path)
        self.update_file_list()

def copy_file(self):
    if hasattr(self, 'selected_item'):
        selected_path = os.path.join(self.current_path, self.selected_item)
        destination = filedialog.askdirectory()
        if destination:
            shutil.copy(selected_path, destination)
            messagebox.showinfo("Копирование", "Файл %s скопирован в %s" % (self.selected_item, destination))

def move_file(self):
    if hasattr(self, 'selected_item'):
        selected_path = os.path.join(self.current_path, self.selected_item)
        destination = filedialog.askdirectory()
        if destination:
            shutil.move(selected_path, destination)
            self.update_file_list()
            messagebox.showinfo("Перемещение", "Файл %s перемещен в %s" % (self.selected_item, destination))
```

Рисунок 4. Методы кнопок программы

Функция “on_select” отвечает за обработку события выбора элемента в дереве. Когда пользователь выбирает файл или папку, этот метод сохраняет имя выбранного элемента в переменной “selected_item”. Это позволяет другим методам использовать выбранный элемент для выполнения различных операций. Функционал “go_back” позволяет пользователю перейти на уровень выше в иерархии директорий. Он проверяет, находится ли текущий путь на корневом уровне, и если нет, переходит к родительской директории, обновляя метку текущего пути и список файлов и папок.

Следующий метод “open_file” открывает выбранный файл или переходит в выбранную папку. Если выбранный элемент является папкой, метод обновляет текущий путь и список файлов и папок. Если это файл, он открывается с помощью системного приложения по умолчанию. Алгоритм “delete_file” удаляет выбранный файл или папку после подтверждения пользователя. Он использует диалоговое окно для подтверждения удаления и затем удаляет элемент, обновляя список файлов и папок.

Операции “create_new_file” и “create_new_folder” позволяют пользователю создавать новые файлы и папки соответственно. Они используют диалоговые окна для ввода имени нового файла или папки и затем создают их в текущей директории, обновляя список файлов и папок. Метод “copy_file” копирует выбранный файл или папку в указанную пользователем директорию. Он использует диалоговое окно для выбора целевой директории и затем выполняет копирование, отображая информационное сообщение о завершении операции.

Функция “move_file” перемещает выбранный файл или папку в указанную пользователем директорию. Он также использует диалоговое окно для выбора целевой директории и затем выполняет перемещение, обновляя список файлов и папок и отображая информационное сообщение о завершении операции. Описанные методы вместе обеспечивают полный

набор функций для управления файлами и папками, делаая файловый менеджер удобным и эффективным инструментом для пользователей.

Разработка программы завершена. После происходит тестирование программы (рис. 5).

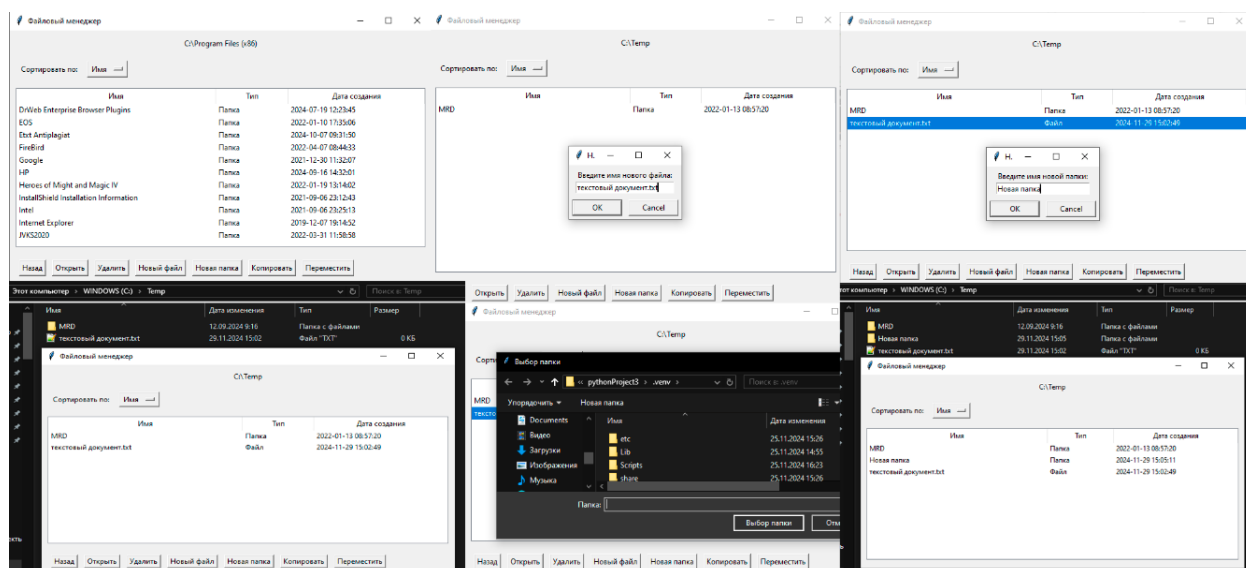


Рисунок 5. Результат тестирования программы

Вывод

Таким образом было разработано графическое приложение, представляющее из себя файловый менеджер. Программа содержит все необходимые функции для работы с папками и файлами. Данная система реализована на языке программирования Python и встроенной библиотеки Tkinter, что говорит о том, что с помощью базовых инструментов выбранного языка можно создавать функциональные программы.

Библиографический список

1. Новичихина А.А. Разработка простых приложений средствами Tkinter в Python // Актуальные исследования. 2023. № 4-1 (134). С. 32-35.
2. Котыбаев Е.С. Разработка пользовательского интерфейса приложения «таймер» на Python // В сборнике: Молодые ученые в решении актуальных проблем науки. Сборник материалов Всероссийской научно-практической конференции студентов, аспирантов и молодых ученых (с международным участием). Красноярск, 2023. С. 635-637.
3. Чернова В.С., Сидорова Е.Б., Орешкина Т.Г. Разработка приложения «Mydiary» на Python // В сборнике: Сборник научных трудов молодых ученых и специалистов. Сборник статей. В 2-х частях. Чебоксары, 2021. С. 408-413.
4. Чудинов К.И. Разработка приложения для обработки и визуализации данных на языке Python // В сборнике: Научное и техническое творчество молодёжи. Материалы Всероссийской научно-технической конференции.

В 4-х частях. Новосибирск, 2024. С. 216-220.

5. Андриенко И.С. Разработка программы для редактирования текста с помощью библиотеки Kivy // Постулат. 2024. № 1 (99). С. 100.