

Разработка интерактивного приложения на языке программирования Python для создания диаграмм Ганта с использованием модулей Tkinter и Matplotlib

Эрдман Александр Алексеевич

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

В данном исследовании рассматривается процесс создания программы, предоставляющей инструменты для построения диаграммы Ганта. Приложение разработано на языке программирования Python, модулей Tkinter и Matplotlib. Результатом исследования является система для создания диаграмм Ганта, а также описание работы программы.

Ключевые слова: python, tkinter, matplotlib

Development of an interactive Python programming language application for creating Gantt charts using Tkinter and Matplotlib modules

Erdman Alexander Alekseevich

Sholom-Aleichem Priamursky State University

Student

Abstract

This study examines the process of creating a program that provides tools for building a Gantt chart. The application is developed in the Python programming language, Tkinter and Matplotlib modules. The result of the research is a system for creating Gantt charts, as well as a description of the program

Keywords: python, tkinter, matplotlib

1 Введение

1.1 Актуальность

Актуальность исследования, тема которого является разработка интерактивного приложения для создания диаграмм Ганта на основе библиотек Tkinter и Matplotlib, обусловлена возрастающей потребностью в эффективных инструментах для управления проектами в различных областях деятельности. В условиях современного динамичного мира бизнеса и технологий, успешное планирование и контроль выполнения задач становятся ключевыми факторами, определяющими успех проектов. Диаграммы Ганта, как визуальный инструмент, позволяют наглядно представлять временные рамки и последовательность выполнения задач, что способствует повышению прозрачности и управляемости проектов. Существующие решения для создания диаграмм Ганта часто ограничены в

функциональности или требуют значительных финансовых вложений, что делает их недоступными для небольших компаний и индивидуальных пользователей. В этом контексте разработка бесплатного и открытого программного обеспечения, использующего популярные и широко распространенные библиотеки Python, такие как Tkinter и Matplotlib, представляет собой оптимальное решение проблемы. Такое приложение позволит пользователям создавать и редактировать диаграммы Ганта без необходимости приобретения дорогостоящих лицензий или изучения специализированных программных продуктов.

1.2 Обзор исследований

Н.А. Иванова и А.В. Свищёв в совместном труде привели описание диаграммы Ганта, объяснили её удобство по сравнению с другой похожей, но менее функциональной структурой: MS Excel [1]. Э.В. Шаповалова и И.В. Свищёва в статье на основе выбранных метрик привели анализ некоторых программных продуктов, обеспечивающих работу с диаграммами Ганта [2]. А.Н. Смагин рассмотрел решение проблемы визуализации данных в Python при помощи библиотеки Matplotlib [3]. Т.А. Халевин и А.А. Голубничий в научной работе представили обзор основ визуализации данных с использованием библиотеки Matplotlib на языке программирования Python [4]. С.В. Радионов привёл описание построения интерфейса для программы, написанной на языке программирования Python [5].

1.3 Цель исследования

Целью исследования является разработка интерактивной программы на языке программирования Python для создания диаграмм Ганта.

2 Материалы и методы

Для реализации программы используется язык программирования Python с модулями Tkinter и Matplotlib. В качестве IDE используется PyCharm.

3 Результаты и обсуждения

Программа начинается с импорта необходимых библиотек и определения основного класса “GanttChartApp”, который инициализирует главное окно приложения и создает основные виджеты (рис. 1). В конструкторе класса “__init__” инициализируются основные переменные и вызывается метод “create_widgets()”, который создает интерфейс приложения. Переменные “tasks”, “duration_days”, “duration_hours” и “duration_minutes” используются для хранения информации о задачах и их продолжительности. Переменные “show_days”, “show_months” и “show_years” управляют отображением временных единиц на диаграмме.

```
import tkinter as tk
from tkinter import ttk, filedialog, messagebox, simpledialog
from tkcalendar import DateEntry
import matplotlib.pyplot as plt
import pickle
from datetime import datetime, timedelta
import matplotlib.dates as mdates

class GanttChartApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Диаграмма Ганта")

        self.tasks = []
        self.duration_days = 0
        self.duration_hours = 0
        self.duration_minutes = 0

        self.show_days = tk.BooleanVar(value=True)
        self.show_months = tk.BooleanVar(value=True)
        self.show_years = tk.BooleanVar(value=True)

        self.create_widgets()
```

Рисунок 1. Основной класс приложения

Следующим шагом является реализация метода “create_widgets” (рис. 2). Данный метод создает две вкладки: “Редактор задач” и “Диаграмма Ганта”. На вкладке “Редактор задач” пользователь может добавлять, редактировать и удалять задачи, а также сохранять и загружать диаграммы. На вкладке “Диаграмма Ганта” отображается сама диаграмма с возможностью настройки отображения временных единиц.

```
def create_widgets(self):
    self.notebook = ttk.Notebook(self.root)
    self.notebook.pack(expand=1, fill="both")

    self.editor_frame = ttk.Frame(self.notebook)
    self.notebook.add(self.editor_frame, text="Редактор задач")

    self.chart_frame = ttk.Frame(self.notebook)
    self.notebook.add(self.chart_frame, text="Диаграмма Ганта")

    self.create_editor_widgets()
    self.create_chart_widgets()
```

Рисунок 2. Создание интерфейса

Для ввода и управления задачами используется функция “create_editor_widgets” (рис. 3).

```
def create_editor_widgets(self):
    self.task_label = tk.Label(self.editor_frame, text="Задача:")
    self.task_label.pack(pady=5)

    self.task_entry = tk.Entry(self.editor_frame)
    self.task_entry.pack(pady=5)

    self.start_label = tk.Label(self.editor_frame, text="Начало:")
    self.start_label.pack(pady=5)

    self.start_entry = DateEntry(self.editor_frame, width=12, background='darkblue',
                                foreground='white', borderwidth=2, date_pattern='yyyy-mm-dd')
    self.start_entry.pack(pady=5)

    self.duration_label = tk.Label(self.editor_frame, text="Продолжительность:")
    self.duration_label.pack(pady=5)

    self.duration_button = tk.Button(self.editor_frame, text="Выбрать продолжительность", command=self.select_duration)
    self.duration_button.pack(pady=5)

    self.add_button = tk.Button(self.editor_frame, text="Добавить задачу", command=self.add_task)
    self.add_button.pack(pady=5)

    self.edit_button = tk.Button(self.editor_frame, text="Редактировать задачу", command=self.edit_task)
    self.edit_button.pack(pady=5)

    self.delete_button = tk.Button(self.editor_frame, text="Удалить задачу", command=self.delete_task)
    self.delete_button.pack(pady=5)

    self.save_button = tk.Button(self.editor_frame, text="Сохранить диаграмму", command=self.save_chart)
    self.save_button.pack(pady=5)

    self.load_button = tk.Button(self.editor_frame, text="Выбрать существующую диаграмму", command=self.load_chart)
    self.load_button.pack(pady=5)

    self.task_listbox = tk.Listbox(self.editor_frame)
    self.task_listbox.pack(pady=5)
```

Рисунок 3. Функционал редактора задач

Этот метод создает метки и поля ввода для названия задачи, даты начала и продолжительности. Кнопки позволяют пользователю выбрать продолжительность, добавить, редактировать и удалить задачи, а также сохранить и загрузить диаграммы. Список задач отображается в виджете “Listbox”.

За настройку продолжительности задач отвечает программный код, находящийся внутри “select_duration” (рис. 4). Данный функционал создает новое окно с полями ввода для дней, часов и минут. После ввода данных и нажатия кнопки “Подтвердить” значения сохраняются в соответствующие переменные, и окно закрывается.

```
def select_duration(self):
    duration_window = tk.Toplevel(self.editor_frame)
    duration_window.title("Выбор продолжительности")

    days_label = tk.Label(duration_window, text="Дни")
    days_label.pack(pady=5)

    days_entry = tk.Entry(duration_window)
    days_entry.pack(pady=5)

    hours_label = tk.Label(duration_window, text="Часы")
    hours_label.pack(pady=5)

    hours_entry = tk.Entry(duration_window)
    hours_entry.pack(pady=5)

    minutes_label = tk.Label(duration_window, text="Минуты")
    minutes_label.pack(pady=5)

    minutes_entry = tk.Entry(duration_window)
    minutes_entry.pack(pady=5)

    def confirm():
        self.duration_days = int(days_entry.get())
        self.duration_hours = int(hours_entry.get())
        self.duration_minutes = int(minutes_entry.get())
        duration_window.destroy()

    confirm_button = tk.Button(duration_window, text="Подтвердить", command=confirm)
    confirm_button.pack(pady=5)

    duration_window.wait_window()
```

Рисунок 4. Настройка продолжительности задачи

Для добавления, редактирования и удаления задач используются методы “add_task”, “edit_task” и “delete_task” (рис. 5).

```

def add_task(self):
    task = self.task_entry.get()
    start = self.start_entry.get_date()
    duration = timedelta(days=self.duration_days, hours=self.duration_hours, minutes=self.duration_minutes)
    self.tasks.append((task, start, duration))
    self.update_task_listbox()
    messagebox.showinfo(title="Ycnex", message=f"Задача '{task}' добавлена.")

def edit_task(self):
    selected_index = self.task_listbox.curselection()
    if not selected_index:
        messagebox.showwarning(title="Предупреждение", message="Выберите задачу для редактирования.")
        return

def delete_task(self):
    selected_index = self.task_listbox.curselection()
    if not selected_index:
        messagebox.showwarning(title="Предупреждение", message="Выберите задачу для удаления.")
        return

```

Рисунок 5. Фрагмент кода функционала редактирования задач

Метод “add_task” добавляет новую задачу в список “tasks”, обновляет список задач в интерфейсе и отображает сообщение об успешном добавлении. Функционал “edit_task” открывает окно для редактирования продолжительности задачи и обновляет информацию о задаче в списке. Для удаления выбранной задачи из списка и обновления списка задач в интерфейсе используется “delete_task”.

Для того, чтобы иметь возможность сохранять созданные диаграммы и в дальнейшем их загружать, применяются функции “save_chart” и “load_chart” (рис. 6).

```

def save_chart(self):
    file_path = filedialog.asksaveasfilename(defaultextension=".pkl", filetypes=[("Pickle files", "*.pkl")])
    if file_path:
        with open(file_path, 'wb') as file:
            pickle.dump(self.tasks, file)
        messagebox.showinfo(title="Ycnex", message="Диаграмма сохранена.")

def load_chart(self):
    file_path = filedialog.askopenfilename(filetypes=[("Pickle files", "*.pkl")])
    if file_path:
        with open(file_path, 'rb') as file:
            self.tasks = pickle.load(file)
        self.update_task_listbox()
        messagebox.showinfo(title="Ycnex", message="Диаграмма загружена.")

```

Рисунок 6. Функции сохранения и загрузки созданных диаграмм

Первая функция открывает диалоговое окно для выбора пути сохранения файла и сериализует список задач “tasks” в файл с использованием библиотеки “pickle”. Вторая функция открывает диалоговое окно для выбора файла и десериализует данные из файла, обновляя список задач в интерфейсе.

Последним реализуется программный код, отвечающий за графическое представление самой диаграммы (рис. 7).



```
def show_chart(self):
    fig, gnt = plt.subplots(figsize=(12, 6))

    gnt.set_xlabel('Время')
    gnt.set_ylabel('Задачи')

    gnt.set_yticks([i for i in range(len(self.tasks))])
    gnt.set_yticklabels([task[0] for task in self.tasks])

    for i, task in enumerate(self.tasks):
        start_date = task[1]
        end_date = start_date + task[2]
        gnt.broken_barh(xranges: [(start_date, end_date - start_date)], yranges: (i - 0.4, 0.6), facecolors='tab:blue')

        gnt.annotate(start_date.strftime('%Y-%m-%d'), xy=(start_date, i + 0.4), xytext=(0, 10),
                    textcoords='offset points', ha='center', va='bottom', fontsize=8, color='black')
        gnt.annotate(end_date.strftime('%Y-%m-%d'), xy=(end_date, i - 0.4), xytext=(0, -10),
                    textcoords='offset points', ha='center', va='top', fontsize=8, color='black')

    min_date = min(task[1] for task in self.tasks)
    max_date = max(task[1] + task[2] for task in self.tasks)
    gnt.set_xlim(min_date - timedelta(days=1), max_date + timedelta(days=1))
    gnt.set_ylim(-1, len(self.tasks))

    if self.show_years.get():
        gnt.xaxis.set_major_locator(mdates.YearLocator())
        gnt.xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
    if self.show_months.get():
        gnt.xaxis.set_minor_locator(mdates.MonthLocator())
        gnt.xaxis.set_minor_formatter(mdates.DateFormatter('%b'))
        plt.setp(gnt.xaxis.get_minorticklabels(), rotation=45, fontsize=8)
    if self.show_days.get():
        gnt.xaxis.set_minor_locator(mdates.DayLocator())
        gnt.xaxis.set_minor_formatter(mdates.DateFormatter('%d'))

    month_names = {
        'Jan': 'Янв', 'Feb': 'Фев', 'Mar': 'Мар', 'Apr': 'Апр', 'May': 'Май', 'Jun': 'Июн',
        'Jul': 'Июл', 'Aug': 'Авг', 'Sep': 'Сен', 'Oct': 'Окт', 'Nov': 'Ноя', 'Dec': 'Дек'
    }
    for label in gnt.xaxis.get_minorticklabels():
        text = label.get_text()
        if text in month_names:
            label.set_text(month_names[text])

    plt.tight_layout()
    plt.show()
```

Рисунок 7. Функционал построения диаграммы

Здесь метод “show_chart” использует библиотеку “Matplotlib” для построения диаграммы Ганта на основе данных о задачах, выполняя несколько ключевых шагов. Сначала создается фигура и оси с заданным размером, а затем устанавливаются метки для осей X и Y, что помогает пользователям понять, что отображается на диаграмме. Далее устанавливаются тики и метки на оси Y, соответствующие каждой задаче, что позволяет легко идентифицировать каждую задачу на диаграмме. Для каждой задачи вычисляются даты начала и окончания, а затем добавляется горизонтальный бар, представляющий продолжительность задачи, с небольшим отступом для лучшей визуализации. Добавляются аннотации для начала и окончания каждой задачи, что позволяет пользователям легко видеть точные даты. Аннотации размещаются с небольшим смещением для лучшей читаемости. Устанавливаются границы для осей X и Y на основе минимальной и максимальной дат начала и окончания задач, что обеспечивает, что все задачи будут видны на диаграмме. В зависимости от выбранных пользователем временных единиц (годы, месяцы, дни), устанавливаются соответствующие локаторы и форматтеры для оси времени, что позволяет гибко настраивать отображение временных меток на диаграмме. Наконец, вызывается “plt.tight_layout()” для автоматической настройки макета диаграммы, чтобы избежать перекрытия элементов, и “plt.show()” для отображения диаграммы в отдельном окне. Таким образом, метод “show_chart” обеспечивает создание наглядной и информативной диаграммы Ганта.

Программа создана и следует её тестирование, в ходе которой демонстрируется работоспособность всего функционала программы (рис. 8).

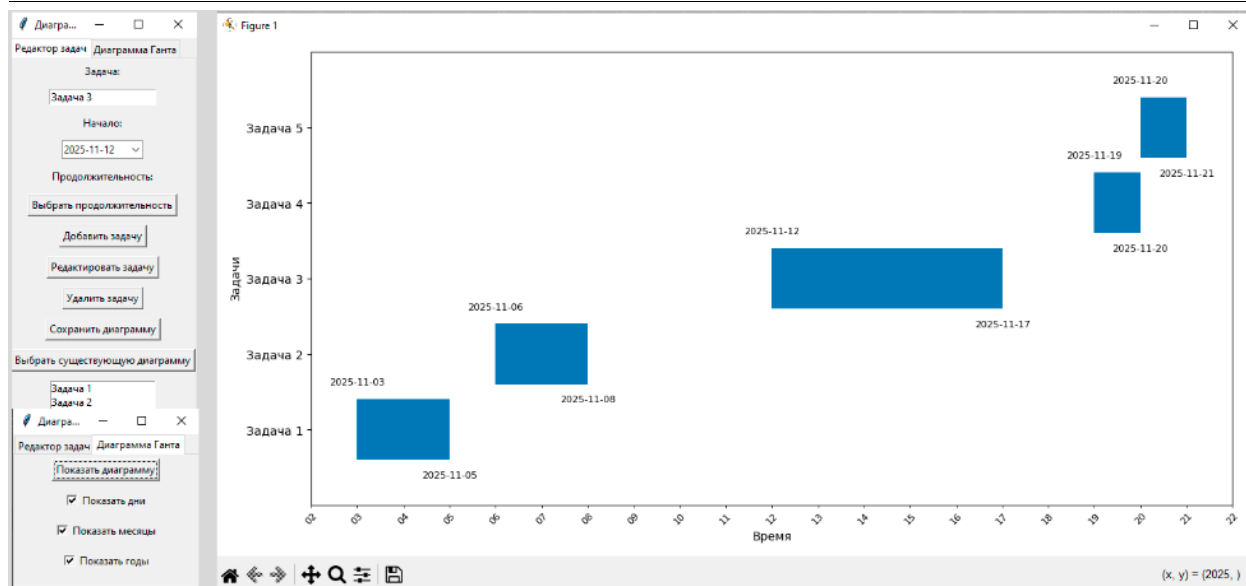


Рисунок 8. Результат работы программы

Вывод

В результате исследования было разработано интерактивное приложение на языке программирования Python для создания и отображения диаграммы Ганта. Программа содержит все базовые инструменты для работы с данной диаграммой. В ходе тестирования системы был получен наглядный пример выполнения поставленной задачи.

Библиографический список

1. Иванова Н.А., Свищёв А.В. Построение диаграммы Ганта в современном мире и ряд преимуществ перед MS Excel // Моя профессиональная карьера. 2022. Т. 2. № 43. С. 166-170.
2. Шаповалова Э.В., Свищёва И.В. Сравнительный анализ программного обеспечения для построения диаграммы Ганта // Моя профессиональная карьера. 2022. Т. 2. № 43. С. 214-220.
3. Смагин А.Н. Визуализация данных в Python при помощи библиотеки Matplotlib // В сборнике: Проблемы управления в социально-экономических и технических системах. Сборник научных статей Материалы XV Международной научно-практической конференции. 2019. С. 39-43.
4. Халевин Т.А., Голубничий А.А. Визуализация данных в Python с использованием библиотеки Matplotlib: основы и примеры // E-Scio. 2023. № 12 (87). С. 26-34.
5. Радионов С.В. Создание интерфейса для программы Python3 с помощью библиотеки Tkinter // Постулат. 2019. № 1-1 (39). С. 41.