ПРИЛОЖЕНИЕ НА TKINTER

URBAN UNIVERSITY | PYTHON

ДЛЯ СОЗДАНИЯ СПИСКА ПОКУПОК НА ОСНОВЕ ВВЕДЕННЫХ РЕЦЕПТОВ И УЖЕ ИМЕЮЩИХСЯ ПРОДУКТОВ

ВАСЯК ВЛАДИМИР ВЛАДИМИРОВИЧ

2025

ДИПЛОМНАЯ РАБОТА НА ТЕМУ:

**«****ПРИЛОЖЕНИЕ НА TKINTER ДЛЯ СОЗДАНИЯ СПИСКА ПОКУПОК**

**НА ОСНОВЕ ВВЕДЕННЫХ РЕЦЕПТОВ И УЖЕ ИМЕЮЩИХСЯ ПРОДУКТОВ.»**

**СОДЕРЖАНИЕ**

***Содержание 2***

***1.Введение 3***

***2.Задачи 4***

***3.Выбор интерфейса 5***

***4.Обзор определений, понятий и методов в области проекта 7***

***5.Этапы разработки 12***

***6.Заключение 16***

1. **ВВЕДЕНИЕ.**

***Почему выбрана именно эта тема:***

В наше время не стоит забывать, что даже обычные повседневные задачи, можно облегчить с помощью программ, которые выполняют за нас те или иные подсчеты и выдают нам нужный результат без затрат времени.

Планируя какой-либо праздник или банкет – всегда нужно накрывать на стол. Однако зачастую очень сложно спланировать точное количество продуктов, которые нужно купить, поскольку во многие блюда нужны одинаковые продукты, иногда получается так, что одного продукта после закупки стало слишком много, несмотря на то что он уже и так был в домашних запасах, а другого наоборот – недостаточно, и приходится снова обдумывать: «Сколько же не хватает?».

1. **ЗАДАЧИ.**

Предлагается создать приложение, которое будет выполнять следующие действия:

* Считывать данные о рецептах и имеющихся продуктах, с указанием их количества.
* Выполнять необходимые проверки и выводить ошибки (при их наличии).
* Выводить «Список покупок», отображая в нем продукты, необходимые для покупки, учитывая все продукты для введенных рецептов и все имеющиеся продукты.

1. **ВЫБОР ИНТЕРФЕЙСА.**

Для решения данной проблемы предлагается написать программу, подсчитывающую точное количество необходимых к покупке продуктов, на основе введенных данных: рецептов и имеющихся продуктов, с указанием количества продуктов в обоих случаях.

Мы выбираем фреймворк «***Tkinter***»

Данный фреймворк обладает некоторым преимуществом перед другими фреймворками благодаря следующему:

1. Он встроен в стандартный пакет Python, то есть ничего дополнительного устанавливать не нужно.
2. Данный фреймворк – кроссплатформенный, то есть на нем можно писать приложения для разных операционных систем.
3. «**Tkinter**» Имеет достаточно широкий встроенный набор интерактивных элементов и атрибутов, что позволяет создать полноценный работающий интерфейс пользователя. Кроме того, есть возможность использовать внешние подключаемые модули, расширяющие возможности пользователей.
4. Его очень просто настраивать и использовать.

Недостатки «**Tkinter»**:

1. Интерфейс данного фреймворка для кого-то покажется устаревшим и древним, несмотря на то что есть возможность улучшать внешний вид с помощью тем и стилей.
2. Набор базовых виджетов и параметров – ограничен, что упрощает работу с фреймворком, а также не затрудняет создание сложных и обширных интерфейсов (что компенсируется возможностью подключать разные модули).
3. Малая производительность. Скорости работы хватает для разработки малых и средних проектов, однако для создания больших – производительность существенно падает.

У данного фреймворка существует достаточно много современных и хороших альтернатив, например – «***PyQt*»**. Но в данном проекте будем говорить именно о фреймворке «***Tkinter***».

Используя фреймворк «***Tkinter***», можно создавать приложения, с которыми легко взаимодействовать пользователям с любым уровнем обращения с компьютером, поэтому, для работы со своей темой я выбрал именно «***Tkinter***».

Также, с помощью этого фреймворка можно создавать интерактивные интерфейсы веб-приложений, сайтов. Он широко используется в тестировании интерфейсов приложений, поскольку на нём можно создать рабочие макеты будущих приложений, для проведения испытаний, которые покажут существующие ошибки и неправильные (и правильные) решения, чтобы в дальнейшем разработать полноценный интерфейс.

Для максимально удобного использования для всех пользователей, мы используем библиотеку-инструмент «**auto-py-to-exe**». Таким образом, чтобы запускать и использовать это приложение, пользователям не потребуется наличие среды **Python** на компьютере.

1. **ОБЗОР ОПРЕДЕЛЕНИЙ, ПОНЯТИЙ И МЕТОДОВ В ОБЛАСТИ ПРОЕКТА.**
2. ***Фреймворк* (Framework) –**

Это набор инструментов и библиотек, упрощающих процесс разработки программного обеспечения. Каждый фреймворк предлагает свою структуру и решения для разных задач, с которыми сталкиваются разработчики. В данном проекте мы рассмотрим фреймворк «**Tkinter»**.

1. ***Класс (Class) –***

Шаблон, с помощью которого легко описываются объекты, в классе содержатся свойства, правила создания и поведение объекта.

1. ***Объект –***

Экземпляр класса, содержащий данные и методы для работы с этими данными.

1. ***Метод –***

Функция, определенная внутри класса, которая определяет его поведение

1. ***Виджет –***

Интерактивный элемент управления. Кнопки, поля текста и т.д.

1. ***Окно (Tk) –***

Основной виджет, группирующий в себе все остальные, является визуальной оболочкой всего интерфейса***.***

1. ***Button (кнопка) –***

Простейший элемент, название говорит само за себя – кнопка – нажатие на которую вызывает определенное действие(функцию).

1. ***Этикетка (Label) –***

Виджет, показывающий некоторый неизменяемый текст, используется для обозначения, указания на действия других виджетов или элементов.

1. ***Поле ввода (Entry) –***

Элемент (конструктор) - представляющий собой поле для ввода однострочного текста, с его помощью можно установить много параметров, основными из них являются:

**background** – фоновый цвет.

**cursor** – курсор указателя мыши.

**foreground** – цвет текста.

**font** – шрифт.

**justify** – выравнивание текста.

**show** – маска вводимых символов.

**state** – для активации и деактивации поля (разрешение изменения) принимает значения NORMAL (по умолчанию) и DISABLED соответственно. (Используется в нашем случае).

**width** – ширина элемента в символах. (Также используется в нашем случае).

**padx –** отступ от границ справа и слева.

**pady –** аналогично **padx**, только сверху и снизу.

Элемент **Entry** обладает некоторыми методами, вот некоторые из них:

* **.insert(index, str)** – позволяет вставить в текстовое поле строку по заданному индексу.
* **.get()** – возвращает введенный текст для дальнейшей работы с ним.
* **.delete(first, last=None)** – удаляет символ с индексом **first**. Если же указать параметр **last** – удаление произойдет до символа с индексом **last**. А чтобы удалить текст до конца – можно задать **last=END**.
* **.focus()** – устанавливает фокус на текстовом поле, которое на данный момент получает ввод с клавиатуры.
* **.strip()** – удаляет лишние пробелы в начале и в конце строки.

1. ***Текст (Text) –***

Виджет, предназначенный для отображения и ввода многострочного текста с возможностью редактирования, обладает всеми параметрами Entry, указанными выше, а также:

**height** – высота виджета в строках.

**wrap –** указание метода переноса текста, если он не умещается в границы виджета. Параметры, которые параметр **wrap** может принимать: **word** (перенос текста по словам), **char** (перенос текста по символам), **none** (без переноса текста).

1. **.pack() –**

Будем использовать данный метод для расположения объектов (виджетов и элементов) в нашем главном окне. Данный метод может обладать различными параметрами:

**expand=True** – заполнение всего окна(контейнера) данным виджетом.

**fill** – определяет, растягивать элемент для заполнения пространства или нет, в зависимости от заданных значений:

* + - * **None** - значение по умолчанию, при нём – элемент не будет растягиваться.
      * **X** – элемент будет растягиваться исключительно по горизонтали.
      * **Y** – элемент будет растягиваться только по вертикали.
      * **BOTH** – элемент будет растягиваться как по вертикали, так и по горизонтали.
* **anchor** – элемент будет размещен в определенной части контейнера: **n**, **s**, **e**, **w**, **nw**, **ne**, **sw**, **se –** что соответствует сторонам света.
* **side** – виджет будет выровнен по одной из сторон окна в зависимости от значения: TOP (верхняя сторона, является значением по умолчанию), RIGHT (правая сторона), BOTTOM (нижняя сторона), LEFT (левая сторона).
* **ipadx** – определяет, каким будет отступ содержимого виджета от его границы по горизонтали.
* **ipady** – определяет, каким будет отступ содержимого виджета от его границы по вертикали.
* **padx –** определяет, каким будет отступ виджета от границ окна по горизонтали.
* **pady –** определяет, каким будет отступ виджета от границ окна по вертикали.

1. ***Окна сообщений (messagebox)*** *–*

Встроенные во фреймворк «**Tkinter**» окна для отображения дополнительных окон с сообщениями, модуль предоставляет следующие функции:

* **showinfo()** – отображает некоторую информацию.
* **showerror()** – отображает ошибки.
* **showwarning()** – отображает предупреждения.
* **title** – заголовок окна.
* **message** – выводимое сообщение.
* **\*\*options** – настройки окна.

1. ***GUI (Графический интерфейс)*** *–*

Оболочка программы, которая включает выше перечисленное, на базе определенной библиотеки (в нашем случае – «**Tkinter»**), а взаимодействие с которой доступно с помощью клавиатуры и мыши.

1. ***Auto-py-to-exe –***

Библиотека Python, позволяющая конвертировать программы **.py** в исполняемые файлы **.exe .**

1. **ЭТАПЫ РАЗРАБОТКИ**

Как говорилось ранее, используем фреймворк «**Tkinter»**. Чтобы начать работу, импортируем его и создаем с его помощью графический интерфейс.

***Подготовка:***

Требуемые библиотеки для работы:

**Tkinter –**

Мы работаем с python3, в которую данная библиотека ходит по умолчанию, поэтому установка не требуется.

**auto-py-to-exe –**

Устанавливаем данную библиотеку: в терминале нашей среды вводим команду: **pip install auto-py-to-exe** и ждем ее завершения.

***Инициализация:***

* Создание окна (**window**), в нем и будет отображаться все наши элементы.
* Создание класса **ShoppingList** для удобства дальнейшей работы. Практически вся работа будет проводиться в этом классе. Класс принимает параметр **window** который и представляет собой основной виджет окна приложения (экземпляр **Tk()** из библиотеки «**Tkinter»**)
* Присваивание имени нашему окну с помощью метода «**.title»**, в нашем случае: «***Список покупок***». А также создание словарей **recipes** для хранения рецептов, **existing\_products** – для хранения имеющихся продуктов
* Создание трех однострочных полей ввода (**Entry),** над каждой из которых создается обозначение в виде метки (**Label**), указывающей на то, для чего нужно каждое поле:
  + - «Название блюда:»
    - «Название ингредиента:»
    - «Количество (г/мл/шт):»

Для удобства использования, при вводе мы не указываем единицу измерения продукта, поскольку пользователь сам понимает в каких единицах измеряется количество того или иного продукта, мы будем использовать универсальное отображение г/мл/шт., что намного удобнее, чем указывать является ли продукт, например, жидкостью или штучным товаром.

* Создание кнопки (**Button**) «***Добавить ингредиент***», выполняющей функцию(метод) **add\_ingredient** (опишем которую позже)
* Создание двух однострочных полей ввода (**Entry**) с указателем в виде метки (**Label**):
  + - «Название продукта:»
    - «Количество (г/мл/шт):»
* Создание кнопки (**Button**) «***Добавить имеющийся продукт***», выполняющей функцию(метод) **set\_existing\_products** (опишем которую также позже).
* Создание кнопки (**Button**) «***Получить список покупок***» выполняющую функцию **get\_shopping\_list**
* Создание многострочного поля (**Text**) для вывода списка покупок.
* Создание многострочного поля (**Text**) с маркировкой (**Label**) «Сохраненные рецепты:» для вывода списка введенных пользователем рецептов.
* Создание многострочного поля (**Text**) с маркировкой (**Label**) «Имеющиеся продукты:» для вывода списка введенных пользователем имеющихся продуктов.

***Функции(методы) описывающие действия кнопок и возвращающие результаты в поля:***

* **add\_ingredient** –

Функция создает для нас тот самый список рецептов (словарь **recipes**): запоминает данные из полей «Название блюда», «Название ингредиента» и «Количество (г/мл/шт)» (первое) проверяет существование рецепта в словаре **recipes** (по названию) если такой есть – добавляет ингредиенты к нему, если нет – создает новый. Также выполняет проверку на заполненность полей, а в поле «Количество (г/мл/шт)» выполняет проверки на принадлежность данных к множеству действительных чисел (float) и является ли оно положительным. Также при успешном выполнении функции (после нажатия на кнопку), поля «Название ингредиента» и «Количество (г/мл/шт)» очищаются, выдается информационное уведомление с помощью модуля **messagebox.showinfo** данные в поле «Сохраненные рецепты:» обновляются функцией **update\_recipes\_area()**, а при ошибках – выдаются уведомления об ошибке с помощью метода **messagebox.showerror.** При отсутствии данных в полях ввода – выводится предупреждение с помощью метода **messagebox.showwarning().**

* **set\_existing\_product –**

Функция заполняет список имеющихся продуктов исходя из введённых нами данных в поля «Название продукта» и «Количество (г/мл/шт)» (второе) проводит аналогичные предыдущей функции проверки, далее при отсутствии ошибок – записывает данные в наш словарь **existing\_products**, выводит его в поле очищает используемые поля ввода, обновляет поле «Имеющиеся продукты», а при наличии ошибок – выводит дополнительное окно с ошибкой с помощью метода **messagebox.showerror.** При отсутствии данных в полях ввода – также выводится предупреждение с помощью метода **messagebox.showwarning().**

* **get\_shopping\_list –**

Функция производит необходимые подсчеты, проверяя, хватает ли нам имеющихся ингредиентов для введённых рецептов, и если их не хватает – добавляет данные в словарь **shopping\_list**, выводит его в поле под кнопкой «Получить список покупок», предварительно очищая его, на случай, если мы изменили или добавили что-то, чтобы избежать повторения. А если же у нас хватает продуктов на введенные рецепты, выводится текст в этом же поле, оповещающий нас о том, что все нужные продукты уже есть.

* **update\_recipes\_area –**

Функция, обновляющая поле «Сохранённые рецепты», дополняя информацию при выполнении функции **add\_ingredient**.

* **update\_existing\_area** –

Функция, обновляющая поле «Имеющиеся продукты», дополняя информацию при выполнении функции **set\_existing\_product**.

Далее создаем объект **app** нашего класса **ShoppingList**, которому передаем наше окно **window**, что позволит нашему классу взаимодействовать с нашим окном.

Теперь превращаем нашу программу в отдельное приложение **.exe**:

В терминале нашей среды запускаем инструмент **auto-py-to-exe**, в открывшемся окне выбираем расположение нашей программы, далее, поскольку наш проект содержит всего один файл - выбираем «**Один файл**». Также выбираем «**Оконное приложение**», чтобы скрыть весь консольный вывод приложения, так как наша программа имеет GUI и далее нажимаем кнопку «**Конвертировать**».

С помощью данного инструмента также можно выбрать иконку приложения, расширения и задать дополнительные опции, но нам этого не требуется.

1. **Заключение**

Мы успешно справились с поставленной задачей в полном объеме. Наше приложение работает так, как и должно. Использовать наше приложение сможет пользователь любого уровня навыков обращения с компьютером.

Библиотека **Tkinter** теперь освоена нами на достаточном уровне, чтобы писать различные приложения малой и средней сложности, что позволит нам повысить свои навыки для выполнения дальнейших задач в области программирования на языке Python.

***Дальнейшие действия***:

Будет полезным научиться красиво оформлять приложения на этой библиотеке, поскольку в наше время очень важно, чтобы приложение было не только легким в обращении, но и красиво выглядело, нравилось и не надоедало.

Для удобства дополнить программу кнопкой копирования текста (сейчас можно копировать только сочетанием клавиш «**ctrl+c»**), а также, при необходимости, кнопкой вывода непосредственно на печать(принтером).

Сгруппировать элементы и виджеты для их удобного отображения в нашем окне.

Также в будущем эту программу можно усовершенствовать до уровня какого-либо продуктового магазина, где вместо рецептов будет вводиться нужное количество тех или иных продуктов.