Spring 2022 CS 687 Capstone Project
Build a Todo List App in Command-Line Interface with Python

Scott Zhou
Advisor: Sion Yoon
MS in Computer Science
School of Technology & Computing (STC)
City University of Seattle (CityU)
zhouscott1@cityuniveristy.edu, yoonhee@cityu.edu

Abstract

Building and using the CLI app can fulfill any users who have desired to use advanced features for multitasking, scripting, automating, and highly customizing compared to the GUI app. CLI has more to offer in scale and better control over system functions with less memory usage. This paper will analyze the reason we choose CLI, what we will be using, and how we build it differently.

Keywords: Python, To-Do App, Command-Line, CRUD, SQLite database

1. INTRODUCTION

Problem Statement

As a developer or an administrator, we are often looking for a lightweight todo app that works across platforms, so it is easy to be used to track daily work and access anywhere via the internet. The CLI app will make this happen. Every computer has a CLI tool, and the project only requires Python to run it. Generally speaking, CLI app makes the development and deployment process much quicker than GUI. CLI app also can be easy to customize by adding additional parameters to the commands that users need to run (Morpheus Data, 2016).

Motivation

CLI has been commonly used for running automated tasks, helpful commands such as ping, gathering information from a file, or executing multiple file system commands. There is always something in CLI that can help user boost their productivity when using a computer or troubleshoot some errors in the operating systems (Morpheus Data, 2016).

Approach

This project covers building a basic CRUD app, using CLI, storing data in the database, and practicing with one of the most popular programming languages Python. This paper will evaluate popular Python modules for building CLI app, and seek a way to improve.

Conclusion

This project produces and delivers quality code in a consistent manner. This allows students to build a basic todo app with computer concepts and programmatic challenges and extend from there to make it work in CLI with advanced features such as storing data in a database or cloud platform. By using Python, the student can focus on development rather than programming syntax and also have more time to work on research in-depth knowledge and ensure the design and system architecture meet user requirements.

2. BACKGROUND

Software act as a bridge to connect hardware components to the users. Generally speaking, as a user, there are two interfaces to interact with an electronic device- GUI and CLI. The name of the interface basically describes how we used them. GUI allows users to interact with a

graphical user interface, and CLI allows users to interact with a command line interface. Users can perform tasks via graphics such as icons and images in GUI and write commands in CLI.

GUI is easy to learn and work with because it is designed to be user-friendly. User getting responses immediately in visualization. On the other hand, users need to have deep knowledge in order to use CLI. New users are more likely to struggle to operate CLI than GUI (Sorzano et al., 2002).

CLI is great at multitasking. Performing multitasking in GUI is not efficient on one screen. In addition, GUI used multiple windows to control, manipulate, and view via folders and programs (Gift & Deza, 2020).

CLI gives users the ability to access and manipulate files on other devices via the network. It is not straightforward without knowing the commands to do so. This is a challenge to new users. GUI allows users to access remotely easier with less experience. A real-world example is IT professionals who manage servers and access user computers remotely (Sampath et al., 2021).

Scripting in CLI requires not only knowing the commands but also scripting syntax. GUI can ease this challenge by using certain programming software to create scripts. There are many examples, including but not limited to compilers, interpreters, and debuggers.

Speed is more important to the professionals who are looking for performance. In GUI, users need to navigate with many different icons, which causes the GUI slow. On the other hand, CLI utilizes the keyboard to navigate instead. This results in faster performance. Many modern GUI has kept trying to improve the speed but still require both mouse and keyboard to interact. At the end of the day, GUI users spend a lot of time taking off their hands from the keyboard in order to move the mouse (Staring & Klein, 2011).

GUI cannot perform advanced tasks as CLI. This is due to CLI having overall control of the whole system and files. Tasks are a lot easier to work within CLI. We can create a script that contains a few command lines to perform most of the tasks. With GUI, users have to manually repeat the actions.

There are some key differences between GUI and CLI: ease of use, multitasking, remote access, scripting, speed, and control. The disadvantage of using CLI is too many commands, even for experienced users. There is no room for mistakes in CLI, mistype a command will not work as expected at all and potentially start over again (Sorzano et al., 2002).

Since GUI is designed for user-friendly, it demands more resources such as memory. Application-based on GUI needs more RAM to run compared to other interface types. A User might need to look up the Help file for hidden commands, and GUI requires more processing power (HT Ling, 2018).

The biggest difference between CLI and GUI is CLI is based on text input, and GUI is based on many graphical components such as icons, menus, and windows. Overall, CLI is more powerful and gives more control with advanced features (Mwikalii, 2021).

3. RELATED WORK

There are many solutions out there for building CLI app. Here are some popular technologies in Python:

- Argparse -Define required arguments in command-line interface.
- Click - Creating modern command line interfaces.
- Typer - easy to start with CLI app.
- Clint - Everything you need for building CLI app from colors, indentation, custom email-style quotes, etc.

Literature Review

An example of building CLI app is to wrap the Machine Learning app into the command-line interface. Google provides the Fire library to transfer a Python program into a command line interface. For an ML app, we can simply have

```
import fire
import pickle
import logging

import pandas as pd
from sklearn.neighbors import KNeighborsClassifier

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

def load_model(model_path: str):
    """Loads model from `model_path`"""

def save_model(model, model_path: str):
    """Saves `model` to `model_path`"""

class Classifier:
    """
    Some classifier, that makes some random classifications.
    """
```

two functions train and predict, and load fire in the main class to make it becomes a CLI app (Balatsko, 2020).

Another example of building CLI app is to use the standard Python library- Argparse. This involves three simple steps, creating ArgumentParser object, adding arguments, and parsing the arguments (Duron, 2021). Here is an example:

```
import argparse
parser = argparse.ArgumentParser(
prog="argparsetest'
description='Test for argparse'
epilog-'This is the end of the test'
argument _default=None,
add help=True)

parser.add_argument (' command',
action=store'
nares='1",
default=None,
choices=('a',
"b', 'c'l.
required-True,
help='File to read')

args = parser.parse args()

if args.command == "a":
do command a()
if args.command == "b"
do command b()
if args. command == "c"
do command c()
```

The third easier option is to use the Click module. This is an alternative choice to the optparse and argparse modules. This module provides arbitrary nesting of commands, auto -generates the help page and tons of lazy loading of subcommands at runtime (Mwaura, 2019).

Review Conclusions

After reviewing and evaluating a few current solutions, I have the following findings:

- There are simple solutions that can convert the CLI app by calling it.
- We can implement the CLI app with Python standard library.
- We need to seek a way to make the interface look more organized and beautiful compared to most of the tutorials.
- Whether or not and how to store the data need to be addressed.

4. APPROACH

Our approach described in this paper for building a todo list CLI app consists of five parts:

1. Use Typer to build CLI applications and create basic features, including create, delete, update, complete and display the work items. Add descriptions in help commands.

2. Use a rich module to support different font colors and styles in the terminal. We will use the table content for our display feature. This module will turn our app into a modern style and not look boring.

3. Create a model for each work item. Possible attributes could be the name of the work item, the category, when it is created and when it is completed, what is the current status, and the positions in the database.

4. Establish a database to match all CRUD features. Check if the table has been created. It will not create one to store all the attributes from the model.

5. Put everything together, load all the database functions into our main python file, and execute all functions corresponding to their command.

How to use this app

The app is really easy to use. Just like any CLI app, you execute the python file and pass the help as an argument to see what are the working commands as shown in Figure 1.

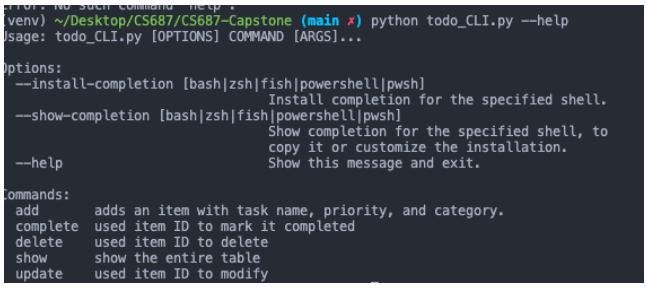

Figure 1

Since our create feature passes Three arguments, when we create a work item we need to give a description, priority and also category. An ID and create date will be auto-generated to the item when we need to update,

mark it as completed, or delete it. Here is an example to create a new work item as show in Figure 2.

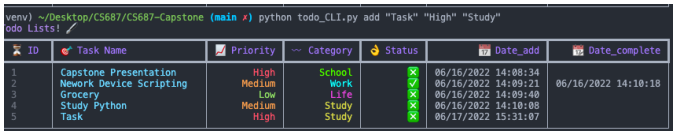

Figure 2

Update command with item ID allow user to update the description, priority and category as shown in Figure 3.

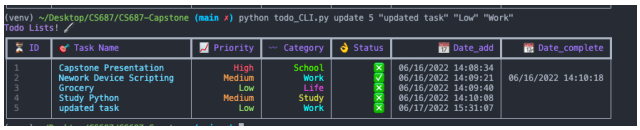

Figure 3

Complete command with item ID will mark the item as completed under status column, also the current date and time will be added under date complete column as shown in Figure 4.

Figure 4

Delete command with item ID will remove unwanted work item from the table as show in Figure 5.

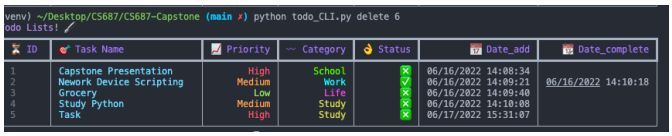

Figure 5

Since we are using Rich module to style our table, it supports various advanced, content, and icons, even emojis. Use the show command to checkout the beautiful styling table in Figure 6.

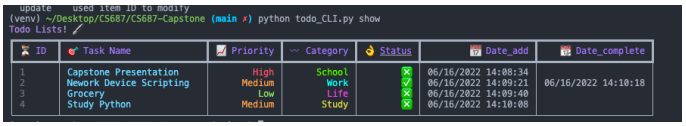

Figure 6

5. DATA COLLECTION

In order to find the bottlenecks in the program, We need to profile them by using the Python edition of Logria, which is powered by curses. In htop command, this will consume almost 100% CPU when it runs (Sardegna, 2021).

All testings are conducted on the MacBook Pro 2019 version with a 1.4 GHz Quad-Core Intel Core i5 and 8GB memory.

We can invoke the cprofile when running the app by writing the following:

```
Python3 -m cProfile -s time todo_CLI.py
```

Cprofile is acting as the entry point for our todo_CLI.py file. The output will look like the result in Figure 7.

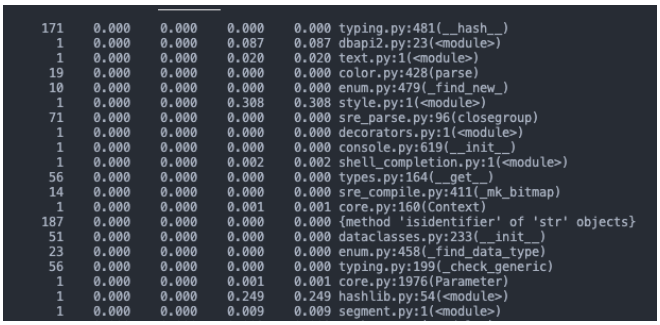

Figure 7 Logria sample output

The columns from left to right are ncalls, tottime, percall, cumtime, and percall respectively. Most informations are related to Python or Curses, and the rest are Logria, for example, shell_output, color_handler, and methods.

6. DATA ANALYSIS

We gather the performance by using the Loria from the previous section. Here are some interesting information worth mentioning:

```
1   0.000   0.000   0.001   0.001 database.py:10(create_table)
1   0.000   0.000   0.091   0.091 database.py:1(<module>)
1   0.000   0.000   0.000   0.000 model.py:5(Todo)
1   0.000   0.000   0.000   0.000 model.py:2(<module>)
1   0.000   0.000   0.007   0.007 main.py:213(__call__)
1   0.000   0.000   0.001   0.001 main.py:217(get_group)
171   0.000   0.000   0.000   0.000 typing.py:481(__hash__)
187   0.000   0.000   0.000   0.000 {method 'isidentifier' of
'str' objects}
212   0.000   0.000   0.000   0.000 {method 'lower' of 'str'
objects}
17   0.000   0.000   0.000   0.000 {method 'reverse' of 'list'
objects}
13   0.000   0.000   0.000   0.000 style.py:22(__init__)
```

Our app runs really fast. There is little room to improve from the performance perspective. The main functions are called once and execute really fast.

Compared to other similar apps: todo.txt, Taskwarrior, or ultralist. Our source code is smaller and provides only the basic feature for a typical todo app. Even someone who has just started writing Python could contribute to our app. The app is minimal to use without any synchronizing, complex features such as setting up a due date or recurrence, and integrating with any API or mobile app.

7. FINDING

The implementation of this CLI todo app because of the need for a lightweight solution for IT professionals that work across different platforms.

The app meets the expectation of being small and fast and also available in a different operating system. This addresses our research question and provides all the functionalities for simple project management in a terminal without any graphical interface. This is intuitive to use and could dramatically improve the workflow.

In order to set up the environment for this app, we only need Python and a few dependency packages to run in the terminal.

The commands are easy to remember after a few practices. The formula is Python3 with our file name, the operation, and optional context if it creates a new item or passes the item ID to modify or delete it. Many commands could be used by them-self without any further inputs. Some commands require additional inputs in order to run properly, for example the add command need user to pass information about the task, all the other commands require users to pass item ID and various options or arguments. Options could be used to modify the core feature of a command, whereas arguments is providing additional information.

Once user inputs a command in the terminal, the CLI app accept it and pass to the shell. The shell converts the commands into actions to be performed by the app. If the commands have output, then the corresponding text will be show in the terminal. Otherwise there will be an error message display instead.

8. CONCLUSION

The proposed research presented in this paper has addressed the problem statement with a positive outcome. We build a lightweight todo app in command line which works across different operating systems. Additionally, this research provides interesting points between GUI and CLI.

This research expects to act as a daily task management solution with the ability to track information such as task ID, task description, categories, and status. The app is built based on CLI, which shortens the implementation time and reduces the complexity even beginner programmers could customize and contribute their own favorite features. The app is easy to set up with only Python and a few of open source dependencies.

The functionality that makes this app stand out from other task management systems is the open source and beginner friendly. Each dependency only calls out the basic usage. The code has been constructed with a traditional model, view, and controller architecture.

This todo app aims to not only provide a quick and simple solution for task managements, but also a great open source project for beginners to start their programming journey. It is highly customizable from the fonts/styles, features and the way how the data is stored.

CLI is like a spoken language, the more users have been used, the better fluency they get.

Many end users ignore the important of command-based system, whereas power users still utilize it to troubleshoot or configure apps or programs. The future of the CLI app may not be highly promoted compare to the latest graphic interfaces, but it is still has its own community with large portion of population. It is definitely worth to learn and practice for any IT professionals.

9. FUTURE WORK

Future research could be adding more features to track more information, such as sub task, due date, assign to, and assign from. In addition, the app only stores data in database, We can add a cloud solution, export to a local file, or sync with a mobile app or browser. Moreover, we anticipate more frequently used features added to this app. We may have to redesign the table in order to avoid too many columns squish ing together. Nevertheless, our results suggest CLI is faster to implement and a must-know skill for IT professionals to improve their daily productivity. Furthermore, ideally for any todo app we need have ensure the following things happened:

1. Adding or organizing tasks should be really fast within a couple taps or keystrokes.
2. We need to offer more than one way to organize these work items.
3. Sending reminder such as notifications, widgets, or emails, so they what needs to be completed before deadline.
4. Keep the UI clean while adding new features is also important to the user experiences.
5. Sync with other platforms for example desktop and mobile.

It is hard to find the perfect apps balances all these things, but since this app is highly customizable, every user can enjoy set it up in their own ways with their own favorite features.

10. REFERENCE

Balatsko, M. (2020, March 16). A simple way to create python CLI app. Towardsdatascience. https://towardsdatascience.com/a-simple-`way-to-create-python-cli-app-1a4492c164b6

Duron, A. (2021, December 13). Making a CLI Application in Python. Medium. https://adrian-td96.medium.com/making-a-cli-application-in-python-a9c7978db54e

Gift, N., & Deza, A. (2020). Python Command Line Tools: Design powerful apps with Click (onemillion2021). Independently published.

HT Ling, M. (2018). RANDOMSEQ: Python command–line random sequence generator. MOJ Proteomics & Bioinformatics, 7(4). https://doi.org/10.15406/mojpb.2018.07.00235

HT Ling, M. (2020). SeqProperties: A Python Command-Line Tool for Basic Sequence Analysis. Acta Scientific Microbiology, 3(6), 103–106. https://doi.org/10.31080/asmi.2020.03.0615

Morpheus Data. (2016, March). Using a CLI for App Development and Deployment. https://morpheusdata.com/cloud-blog/using-a-cli-for-app-development-and-deployment

Mwaura, M. (2019, July 2). Create Your First CLI Application With Python Click. Medium. https://betterprogramming.pub/python-click-building-your-first-command-line-interface-application-6947d5319ef7

Mwikalii, B. (2021, April 6). Comparing Graphical User Interface (GUI) and Command Line Interface (CLI). Section.Io. https://www.section.io/engineering-education/comparing-graphical-user-interface-gui-and-command-line-interface-cli/

Sampath, H., Merrick, A., & Macvean, A. (2021). Accessibility of Command Line Interfaces. https://doi.org/10.1145/3411764.3445544

Sardegna, C. (2021, February 8). Making Python CLI Apps Faster. Chrissardegna. https://chrissardegna.com/blog/making-python-cli-apps-faster/

Sorzano, C. O. S., Carazo, J. M., & Trelles, O. (2002). Command-line interfaces can be efficiently brought to graphics: COLIMATE (the COmmand LIne MATE). Software: Practice and Experience, 32(9), 873–887. https://doi.org/10.1002/spe.465

Staring, M., & Klein, S. (2011). A Simple Command Line Argument Parser. The Insight Journal. https://doi.org/10.54294/t58bxn