Creating a CLI application in Python

Introduction

Command-line interfaces (CLIs) are a powerful way to interact with software. They allow users to run programs and perform tasks by typing commands into a terminal or console. CLIs are often used for automation, scripting, and system administration tasks, but they can also be used to create user-friendly interfaces for complex software.

Hello, user!

This first example will accept a name as input and print a greeting message to the user.

We will use the `argparse` module from the Python standard library to parse the command-line arguments. This module makes it easy to create command-line interfaces for your Python programs.

The first step is to import the `argparse` module and create an `ArgumentParser` object. This is where we define the command-line arguments that our program will accept.
In this case we only accept one argument, `name`, which is a string.
We can then access the value of the `name` argument using the `args` object that `argparse` creates for us.

```python
import argparse

parser = argparse.ArgumentParser(description="Print a greeting message to the user.")
parser.add_argument("name", type=str, help="The name of the user to greet.")

args = parser.parse_args()

print(f"Hello, {args.name}!")
```

We can now save the file, for example as CLI.py and run it as

```
python CLI.py Nico
```

Which will output

```
Hello, Nico!
```

What if we don't provide a name? We get an error

```
usage: CLI.py [-h] name
CLI.py: error: the following arguments are required: name
```

A more complex example

In this second example we will create an application that will take a list of numbers as input and return either the sum, the mean, or the maximum of the numbers.

In this case we define two positional arguments: `numbers`, which is a list of numbers, and two optional arguments: `--sum` and `--mean`, which specify whether to calculate the sum, the mean or the maximum of the numbers (or any combination of these).

Note how we use `nargs="+"` to specify that the `numbers` argument can take one or more values. Also, we use `action="store_true"` to specify that the `--sum`, `--mean`, and `--max` arguments do not take any values, but are simply flags that are either present or not. If we include `--sum` in the command line, the `args.sum` attribute will be `True`, otherwise it will be `False`.

```python
import argparse

parser = argparse.ArgumentParser(description="Calculate the sum, mean, or median 
of a list of numbers.")
parser.add_argument("numbers", type=float, nargs="+", help="List of numbers to 
calculate the sum, mean, or median of.")
parser.add_argument("--sum", action="store_true", help="Calculate the sum of the 
numbers.")
parser.add_argument("--mean", action="store_true", help="Calculate the mean of the 
numbers.")
parser.add_argument("--max", action="store_true", help="Calculate the maximum of 
the numbers.")

args = parser.parse_args()

if args.sum:
    print(f"The sum of the numbers is: {sum(args.numbers)}")

if args.mean:
    print(f"The mean of the numbers is: {sum(args.numbers) / len(args.numbers)}")

if args.max:
    print(f"The maximum of the numbers is: {max(args.numbers)}")

# Because we have defined the --sum, --mean, and --max arguments as optional, we 
need to check if at least one of them is present.
if not any([args.sum, args.mean, args.max]):
    print("No operation specified. Please use --sum, --mean, or --max.")
```