**Основы интерфейса командной строки**

Интерфейс командной строки (CLI) начинается с имени исполняемого файла. Введя его имя в консоли, пользователь получает доступ к главной точке входа в скрипт, например, к **pip**.

Основными параметрами передаваемые в CLI приложение можно разделить на следующие группы:

* **Аргументы** – обязательные параметры, передаваемые скрипту. Если их не определять, CLI вернет ошибку. Например, django– это аргумент в команде pip install django.
* **Опции** – необязательные ([]) параметры, объединяющие имя и часть значения, например -cache-dir ./my-cache. Программе pip сообщается, что значение ./my-cache должно использоваться для определения каталога кэша.
* **Флаги** – специальные опции, которые включают или отключают определенное поведение. Чаще всего это --help.

Вероятно, вы уже использовали CLI, когда устанавливали Python пакет, выполнив pip install <PACKAGE NAME>. Команда install сообщает CLI, что необходимо получить доступ к функции устанавливающей пакет и предоставить доступ к параметрам, характерным для этой функции.

## Фреймворки командной строки,

## доступные в стандартной библиотеке Python 3.x

Добавление команд и опций в скрипты чрезвычайно полезно, но разбор командной строки не так очевиден, как это может показаться. Вместо того, чтобы начинать писать свои собственные парсеры, рационально воспользоваться одним из множества Python пакетов, которые уже решили эту задачу.

Двумя наиболее известными пакетами являются optparse и argparse. Они являются частью стандартной библиотеки Python по принципу «батарейки идут в комплекте».

Они в основном обеспечивают схожую функциональность и работают очень похоже. Самое большое отличие заключается в том, что библиотека optparse устарела и запрещена в Python 3.2, а argparse считается стандартом для реализации CLI в Python.

Более подробную информацию о них можно найти в документации по Python.

Однако при использовании вышеупомянутых библиотек код нельзя назвать интуитивно понятным и легко читаемым, поэтому в данном примере используется альтернатива – библиотека **Click**.

<https://digitology.tech/posts/napisanie-python-utilit-komandnoj-stroki-s-click/>

**Click** разрешает ту же задачу, что и optparse и argparse, но использует несколько иной подход. Он использует концепцию декораторов. Это требует реализацию команд в виде функций, которые потом можно обернуть декоратором.

Например, простое CLI приложение на основе **Click**, который распечатывает текст в консоль выглядит так:

# cli.py

**import** click

@click.command()

**def** main():

print("Hello, World!")

**if** \_\_name\_\_ == "\_\_main\_\_":

main()

Всё что нужно сделать – это создать функцию и добавить к ней декоратор @click.command(). Он превращает функцию main в команду **Click**, которая является точкой входа для скрипта. После запуска скрипта получим результат:

$ python cli.py

Hello, World!

Прелесть click – это то, что мы бесплатно получаем некоторые дополнительные функции. Например, не реализовывая никаких вспомогательных возможностей добавив параметр

--help при запуске скрипта, распечатается основная страница справки в командную строку:

$ python cli.py --help

Usage: cli.py [OPTIONS]

Options:

--help Show this message and exit.

Описание более сложного примера (взаимодействию с OpenWeatherMap API) можно найти по следующей ссылке:

<https://digitology.tech/posts/napisanie-python-utilit-komandnoj-stroki-s-click/>

В коде примера к лабораторной работе также используются декораторы @cli.command для обьявления команд, не принимающих параметры, а также @click.option для объявления дополнительных параметров, необходимых для выполнения конкретной команды.