

ХОЧУ ПОМОЧЬ
ПРОЕКТУ

gb.ru

РЕКЛАМА • 16+

Бесплатный практикум для детей: Python и анимация

Живой практикум для детей по 2D-анимации и Python. Количество мест ограничено.

Узнать больше

Страница 3. / Модуль click в Python, создание CLI интерфейсов

Сообщения

Модуль click в Python, создание CLI интерфейсов

коллекция

Онлайн-презентация

Преимущества

- Простота использования.
- Поддержка параметров командной строки.
- Автоматическая загрузка подкоманд во время выполнения.
- Сравнение с argparse.

Дополнительно

12+. Сроки проведения с 18.09.23 по 30.09.23. Товары и условия их распродажи определяются продавцами. Также товары и отменены знаком «Распродажа». Представлены собирательные образы товара.

Модуль click в Python, создание CLI интерфейсов

одной строки на Python

он для создания красивых интерфейсов командной строки компонуемым способом с минимальным количеством кода. Он легко настраивается, но по умолчанию поставляется с разумными настройками. Это делает процесс написания инструментов командной строки быстрым и увлекательным, а также позволяет избежать проблем, вызванных невозможностью реализовать предполагаемый API CLI.

Модуль click в Python, создание CLI интерфейсов

анд.

правки по параметрам командной строки.

агрузку подкоманд во время выполнения.

о сравнению с argparse.

т командной строки из имеющихся функций можно использовать [сторонний модуль fire](#).

Почему именно click, а не встроенный модуль argparse?

[Модуль argparse](#) имеет некоторые особенности поведения, которые затрудняют обработку произвольных интерфейсов командной строки:

- argparse имеет встроенное поведение, которое пытается угадать, является ли что-то параметром или опцией. Такое поведение становится непредсказуемым при работе со сценариями, в которых не используется часть опций и/или параметров.
- argparse не поддерживает отключение перемежающихся аргументов. Без этой функции невозможно безопасно реализовать вложенный синтаксический анализ, например как в click.

Установка модуля click в виртуальное окружение.

```
# создаем виртуальное окружение, если нет
$ python3 -m venv .venv --prompt VirtualEnv
# активируем виртуальное окружение
$ source .venv/bin/activate
# ставим модуль click
(VirtualEnv):~$ python3 -m pip install -U click
```

Содержание:

- [Простой пример сценария с модулем click](#);
 - [Модуль click и пакет setuptools](#);
- Базовые концепции модуля click;
 - [Создание команды](#);
 - [Функция click.echo\(\)](#);
 - [Вложенные команды](#);
 - [Отложенная регистрация подкоманд](#);
 - [Добавление параметров командной строки](#).

Простой пример сценария с модулем click:

```
@click.command()
@click.option('--count', default=1, help='Number of greetings.')
@click.option('--name', prompt='Your name', help='The person to greet.')
def hello(count, name):
    """Приветствует ИМЯ (`name`), несколько (`count`) раз."""
    РЕКЛАМА
    for x in range(count):
        click.echo(f"Hello {name}!")

if __name__ == '__main__':
    hello()
```

Если запустить эту программу в командной строке то вывод будет следующим:

```
$ python hello.py --count=3
# Your name: John
# Hello John!
# Hello John!
# Hello John!
```

К тому же программа, на основе модуля click автоматически генерирует красивые справочные страницы:

```
$ python hello.py --help
# Usage: hello.py [OPTIONS]
#
#   Simple program that greets NAME for a total of COUNT times.
#
# Options:
#   --count INTEGER  Number of greetings.
#   --name TEXT       The person to greet.
#   --help            Show this message and exit.
```

Модуль click и пакет setuptools.

В коде программы, есть блок в конце файла, который выглядит следующим образом: `if __name__ == '__main__' :.` Традиционно так выглядит автономный файл Python, но есть способ сделать использование написанной утилиты командной строки лучше и проще с помощью инструментов |setuptools|.

Для этого есть две основные (и многие другие) причины:

Во-первых, setuptools автоматически генерирует исполняемые оболочки для Windows, следовательно утилиты командной строки работают и в Windows.

Вторая причина заключается в том, что сценарии setuptools работают с [virtualenv](#) в Unix без необходимости активации virtualenv. Это очень полезная концепция, которая позволяет объединить написанные скрипты со всеми зависимостями в виртуальную среду virtualenv.

Дополнительную информацию смотрите в разделе "[Интеграция модуля click с setuptools](#)".

Базовые концепции модуля click.

Создание команды.

Модуль click основан на объявлении команд через [декораторы](#). Внутри модуля есть интерфейс без декоратора для сложных случаев использования, но он не рекомендуется для высокоуровневого кода.

Функция становится инструментом командной строки, если она декорируется с помощью [@click.command\(\)](#). В самом простом случае, если просто украсить функцию этим декоратором, то она превратится в вызываемый скрипт:

```
# hello.py
import click

@click.command()
def hello():
    click.echo('Hello World!')

if __name__ == '__main__':
    hello()
```

Примечание: то, что декоратор `@click.command()` преобразует функцию в команду, которая затем может быть вызвана:

Вверх

```
$ python hello.py
# Hello World!

# И соответствующая страница помощи
$ python hello.py --help
# Usage: hello.py [OPTIONS]
#
# Options:
#   --help  Show this message and exit.
```

Функция click.echo().

Почему в этом примере используется [функция click.echo\(\)](#) вместо обычной [функции print\(\)](#)? Ответ на этот вопрос заключается в том, что Модуль click пытается последовательно поддерживать различные среды и быть очень надежным, даже если среда настроена неправильно. Click спроектирован, что-бы быть функциональным, по крайней мере, на базовом уровне, даже если все полностью сломано.

Это означает, что функция click.echo() применяет некоторое исправления ошибок в случае, если кодировка терминала настроена неправильно.

Функция click.echo() также поддерживает цвет и другие стили вывода. Она автоматически удалит стили, если выходной поток является файлом. В Windows автоматически устанавливается и используется модуль colorama.

Вложенные команды.

Для простых сценариев командной строки можно автоматически присоединить и создать подкоманду, при помощи [декоратора @click.group\(\)](#).

```
import click

# создаем группу команд `cli`
@click.group()
def test():
    pass

# обратите внимание на название
# декораторов для вложенных команд

# присоединяем команду `initdb`
@test.command()
def initdb():
    click.echo('Initialized the database')

# присоединяем команду `dropdb`
@test.command()
def dropdb():
    click.echo('Dropped the database')

if __name__ == '__main__':
    test()
```

В примере выше, декоратор [@click.group\(\)](#) расположен над основной функцией/командой test() и создает объект Group с именем этой функции test, которому можно дать несколько подкоманд. Подкоманды украшаются декораторами @group.command(), в примере, это декоратор с именем @test.command().

Отложенная регистрация подкоманд.

Вместо использования декоратора @group.command(), подкоманды могут быть украшены простым декоратором @click.command() и позже зарегистрированы в группе при помощи group.add_command(). Такое поведение может быть использовано для разделения подкоманд на несколько модулей Python.

```
import click

# создаем команду, с именем `initdb`
@click.command()
def initdb():
    click.echo('Initialized the database')

# 

Вверх

 команду, с именем `dropdb`
```

12.09.2023, 15:46

Модуль click в Python, создание CLI интерфейсов

```
@click.command()
def dropdb():
    click.echo('Dropped the database')

# создаем группу команд, с именем `cli`
@click.group()
def test():
    pass

# добавляем `initdb` и `dropdb` как подкоманды сценария `test`
test.add_command(initdb)
test.add_command(dropdb)

if __name__ == '__main__':
    test()
```

Добавление параметров командной строки.

Чтобы добавить параметры командной строки к сценарию, необходимо использовать декораторы [@click.option\(\)](#) и [@click.argument\(\)](#):

```
# hello.py
import click

@click.command()
@click.option('--count', default=1, help='number of greetings')
@click.argument('name', help='You name')
def hello(count, name):
    """
    This script prints "Hello <NAME>!" COUNT times.

    - <NAME> is your name.
    """
    for x in range(count):
        click.echo(f"Hello {name}!")

if __name__ == '__main__':
    hello()
```

Запускаем сценарий:

```
$ python hello.py --count=3 John
# Hello John!
# Hello John!
# Hello John!
```

Теперь с опцией --help:

```
$ python hello.py --help
# Usage: hello.py [OPTIONS] NAME
#
#   This script prints "Hello <NAME>!" COUNT times.
#
#   - <NAME> is your name.
#
# Options:
#   --count INTEGER  number of greetings
#   --help           Show this message and exit.
```

Содержание раздела:

- [КРАТКИЙ ОБЗОР МАТЕРИАЛА.](#)
- [Интеграция модуля click с setuptools](#)
- [Стиль и цвета при выводе текста в терминал, модуль click](#)
- [Опции сценария командной строки модуля click](#)
- [Позиционные параметры командной строки модуля click](#)
- [Вверх](#) [ые типы опций и параметров модуля click](#)
- [Произвольное вложение команд в сценариях модуля click](#)

- [Запрос на ввод данных, подтверждение действий в сценариях модуля click](#)
- [Настройка страницы справки сценария на click](#)
- [Индикатор выполнения для модуля click](#)
- [Прокрутка длинного текста в терминале с модулем click](#)
- РЕКЛАМА [Ожидание нажатия клавиши в сценарии click](#)
- [Запуск приложений ОС из сценария на click](#)