

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ
ФЕДЕРАЦИИ**
**Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВЕРОКАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

Кафедра инфокоммуникаций

Институт цифрового развития

ОТЧЁТ

по лабораторной работе №2.17

Дисциплина: «Программирование на Python»

**Тема: «Разработка приложений с интерфейсом командной строки
(CLI) в Python3»**

Выполнил: студент 2 курса

группы ИВТ-б-о-21-1

Кочкаров Умар Ахматович

Ставрополь 2022


Цель работы: приобретение построения приложений с интерфейсом командной строки с помощью языка программирования Python версии 3.x.

Ход работы:

1. Создать общедоступный репозиторий с лицензией MIT и языком Python.

Owner *

Repository name *


 umarkochkarov ▾

 /


lb2.17 ✓

Great repository names are short and memorable. Need inspiration? How about [musical-lamp?](#)

Description (optional)

☒  **Public**

Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**

You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

☒ **Add a README file**


This is where you can write a long description for your project. [Learn more.](#)


Add .gitignore
Choose which files not to track from a list of templates. [Learn more.](#)

.gitignore template: Python ▾

Choose a license
A license tells others what they can and can't do with your code. [Learn more.](#)

License: MIT License ▾

This will set  **main** as the default branch. Change the default name in your [settings](#).

 You are creating a public repository in your personal account.

Create repository

Рисунок 1. Создание репозитория

2. Клонировать репозиторий на ПК:

```
erken@LAPTOP-ESTC60GF MINGW64 ~/Desktop/python/Ла62.17
$ git clone https://github.com/umarkochkarov/lb2.17.git
Cloning into 'lb2.17'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), done.
```

Рисунок 2. Клонирование репозитория

3. Организовать репозиторий в соответствии с моделью ветвления git-flow.

```
erken@LAPTOP-ESTC60GF MINGW64 ~/Desktop/python/Ла62.17/lb2.17 (main)
$ git flow init

Which branch should be used for bringing forth production releases?
- main
Branch name for production releases: [main]
Branch name for "next release" development: [develop]

How to name your supporting branch prefixes?
Feature branches? [feature/]
Bugfix branches? [bugfix/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? []
Hooks and filters directory? [C:/Users/erken/Desktop/python/Ла62.17/lb2.17/.git/hooks]
```

Рисунок 3. Организация репозитория в соответствии с моделью git-flow

4. Проработка примеров из лабораторной работы:

```
C:\Users\erken\Desktop\python\Ла62.17\lb2.17\prim>python primer.py add data.json --name="Иван Иванов" --post="Главный инженер" --year=2012

C:\Users\erken\Desktop\python\Ла62.17\lb2.17\prim>primer.py display data.json
+-----+-----+-----+
|          Ф.И.О.          |    Должность    |   Год   |
+-----+-----+-----+
1 | Иван Иванов              | Главный инженер |   2012  |
+-----+-----+-----+

C:\Users\erken\Desktop\python\Ла62.17\lb2.17\prim>primer.py select data.json --period=10
+-----+-----+-----+
|          Ф.И.О.          |    Должность    |   Год   |
+-----+-----+-----+
1 | Иван Иванов              | Главный инженер |   2012  |
+-----+-----+-----+
```

Рисунок 4. Пример из лабораторной работы

5. Индивидуальные задания

1) Задание: для своего варианта лабораторной работы 2.16 необходимо дополнительно реализовать интерфейс командной строки (CLI).

```
C:\Users\erken\Desktop\python\Ла62.17\lb2.17\ind>python ind1.py add ind1.json --name="Trrain" --no=21 --time="15:00:00"
C:\Users\erken\Desktop\python\Ла62.17\lb2.17\ind>python ind1.py display ind1.json
```

No	Название	Время
21	Trrain	15:00:00

Рисунок 5. Индивидуальное задание 1

- 2) Задание повышенной сложности: самостоятельно изучите работу с пакетом `click` для построения интерфейса командной строки (CLI). Для своего варианта лабораторной работы 2.16 необходимо реализовать интерфейс командной строки с использованием пакета `click`

```
S C:\Users\super\Desktop\Лина\ВУЗ\Программирование на python\Lab2_17\prog> python individual12.py -c display ind1.json
```

No	Название	Время
1	One	12:21:00
2	Two	12:21:00
3	Three	21:21:00

Рисунок 6. Индивидуальное задание 2

Контрольные вопросы:

1. В чем отличие терминала и консоли?

Терминал (от лат. *terminus* — граница) — устройство или ПО, выступающее посредником между человеком и вычислительной системой. Обычно данный термин используется, когда точка доступа к системе вынесена в отдельное физическое устройство и предоставляет свой пользовательский интерфейс на основе внутреннего интерфейса (например, сетевых протоколов). Консоль *console* — исторически реализация терминала с клавиатурой и текстовым дисплеем. В настоящее время это слово часто используется как синоним сеанса работы или окна оболочки командной строки. В том же смысле иногда применяется и слово “терминал”.

2. Что такое консольное приложение?

Консольное приложение *console application* — вид ПО, разработанный с расчётом на работу внутри оболочки командной строки, т.е. опирающийся на текстовый ввод-вывод.

3. Какие существуют средства языка программирования Python для построения приложений командной строки?

Python 3 поддерживает несколько различных способов обработки аргументов командной строки. Встроенный способ – использовать модуль `sys`. С точки зрения имен и использования, он имеет прямое отношение к библиотеке C (`libc`). Второй способ – это модуль `getopt`, который обрабатывает как короткие, так и длинные параметры, включая оценку значений параметров. Кроме того, существуют два других общих метода. Это модуль `argparse`, производный от модуля `optparse`, доступного до Python 2.7. Другой метод – использование модуля `docopt`, доступного на GitHub. У каждого из этих способов есть свои плюсы и минусы, поэтому стоит оценить каждый, чтобы увидеть, какой из них лучше всего соответствует вашим потребностям.

4. Какие особенности построение CLI с использованием модуля `sys`?

Это базовый модуль, который с самого начала поставлялся с Python. Он использует подход, очень похожий на библиотеку C, с использованием `argc` и `argv` для доступа к аргументам. Модуль `sys` реализует аргументы командной строки в простой структуре списка с именем `sys.argv`. Каждый элемент списка представляет собой единственный аргумент. Первый элемент в списке `sys.argv` [0] – это имя скрипта Python. Остальные элементы списка, от `sys.argv` [1] до `sys.argv` [n], являются аргументами командной строки с 2 по n. В качестве разделителя между аргументами используется пробел. Значения аргументов, содержащие пробел, должны быть заключены в кавычки, чтобы их правильно проанализировал `sys`. Эквивалент `argc` – это просто количество элементов в списке. Чтобы получить это значение, используйте оператор `len()`. Позже мы покажем это на примере кода.

5. Какие особенности построение CLI с использованием модуля `getopt`?

Как вы могли заметить ранее, модуль `sys` разбивает строку командной строки только на отдельные фасы. Модуль `getopt` в Python идет немного дальше и расширяет разделение входной строки проверкой параметров.

Основанный на функции C `getopt`, он позволяет использовать как короткие, так и длинные варианты, включая присвоение значений.

6. Какие особенности построение CLI с использованием модуля `argparse`?

Для начала рассмотрим, что интересного предлагает `argparse`:

- анализ аргументов `sys.argv`;
- конвертирование строковых аргументов в объекты Вашей программы и работа с ними;

- форматирование и вывод информативных подсказок. Одним из аргументов противников включения `argparse` в Python был довод о том, что в стандартных модулях и без этого содержится две библиотеки для семантической обработки (парсинга) параметров командной строки. Однако, как заявляют разработчики `argparse`, библиотеки `getopt` и `optparse` уступают `argparse` по нескольким причинам:

- обладая всей полнотой действий с обычными параметрами командной строки, они не умеют обрабатывать позиционные аргументы (`positional arguments`). Позиционные аргументы — это аргументы, влияющие на работу программы, в зависимости от порядка, в котором они в эту программу передаются. Простейший пример — программа `cp`, имеющая минимум 2 таких аргумента («`cp source destination`»).

- `argparse` дает на выходе более качественные сообщения о подсказке при минимуме затрат (в этом плане при работе с `optparse` часто можно наблюдать некоторую избыточность кода);

- `argparse` дает возможность программисту устанавливать для себя, какие символы являются параметрами, а какие нет. В отличие от него, `optparse` считает опции с синтаксисом наподобие `"-pf, -file, +rgb, /f` и т.п. «внутренне противоречивыми» и «не поддерживается `optpars` 'ом и никогда не будет»;

- `argparse` даст Вам возможность использовать несколько значений переменных у одного аргумента командной строки (`nargs`);

- `argparse` поддерживает субкоманды (`subcommands`). Это когда основной парсер отправляет к другому (субпарсеру), в зависимости от аргументов на входе.

Вывод: в результате выполнения работы были приобретены знания о построении приложений с интерфейсом командной строки с помощью языка программирования Python версии 3.x