

**РОССИЙСКОЙ ФЕДЕРАЦИИ**  
**Федеральное государственное автономное**  
**образовательное учреждение высшего образования**  
**«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

**Кафедра инфокоммуникаций**  
**«Разработка приложений с интерфейсом командной строки (CLI) в**  
**Python3»**

**Отчет по лабораторной работе № 2.17**  
**по дисциплине «Программирование на Python»**

Выполнил студент группы ИВТ-б-о-21-1

Шайдеров Дмитрий Викторович.

«21» ноября 2022г.

Подпись студента \_\_\_\_\_

Работа защищена «    » \_\_\_\_\_ 20\_\_ г.

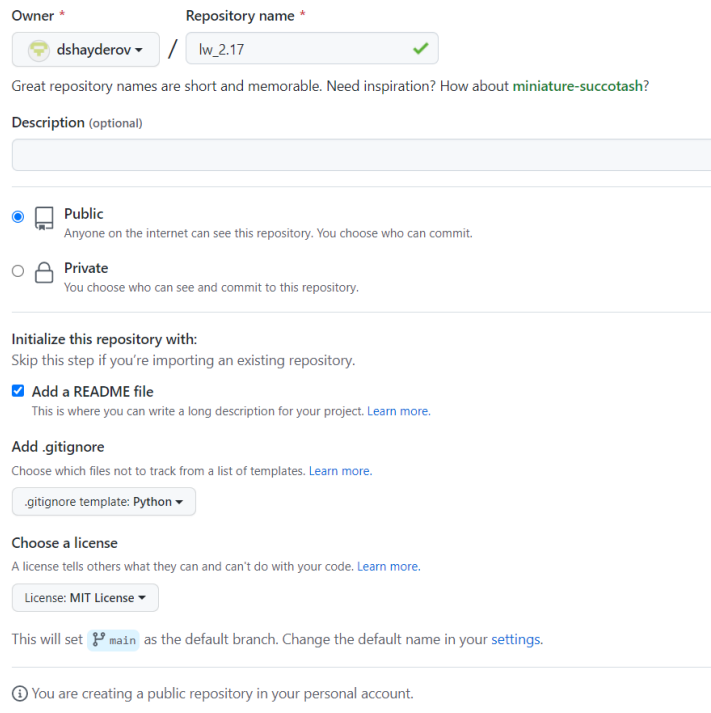
Проверил Воронкин Р.А. \_\_\_\_\_  
(подпись)

Ставрополь 2022

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

**Порядок выполнения работы:**

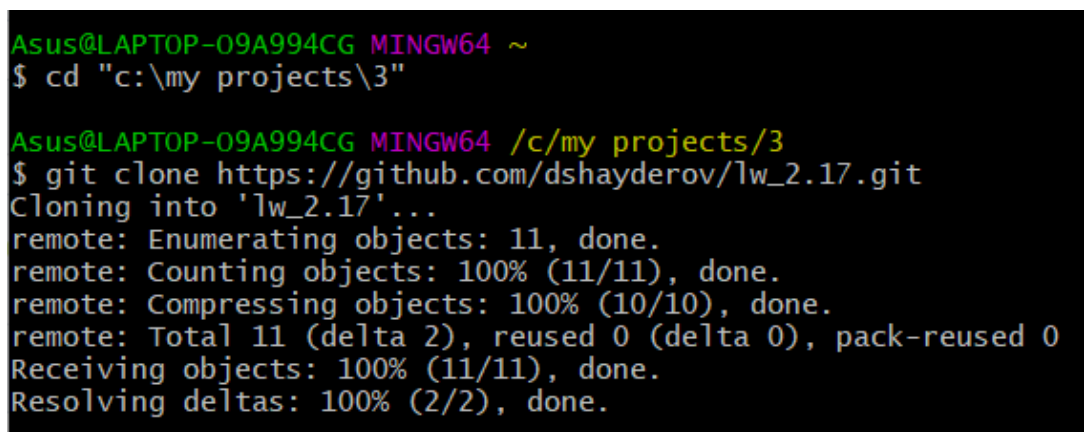
1. Создал общедоступный репозиторий на GitHub, в котором использована лицензия MIT и язык программирования Python.



The screenshot shows the GitHub 'Create new repository' form. The 'Owner' is 'dshayderov' and the 'Repository name' is 'lw\_2.17'. The 'Description' field is empty. The 'Public' option is selected. Under 'Initialize this repository with:', the 'Add a README file' checkbox is checked. The '.gitignore' template is set to 'Python'. The 'License' is set to 'MIT License'. A note at the bottom states: 'You are creating a public repository in your personal account.'

Рисунок 1 - Создание репозитория

2. Выполните клонирование созданного репозитория.



```
Asus@LAPTOP-09A994CG MINGW64 ~  
$ cd "c:\my projects\3"  
  
Asus@LAPTOP-09A994CG MINGW64 /c/my projects/3  
$ git clone https://github.com/dshayderov/lw_2.17.git  
Cloning into 'lw_2.17'...  
remote: Enumerating objects: 11, done.  
remote: Counting objects: 100% (11/11), done.  
remote: Compressing objects: 100% (10/10), done.  
remote: Total 11 (delta 2), reused 0 (delta 0), pack-reused 0  
Receiving objects: 100% (11/11), done.  
Resolving deltas: 100% (2/2), done.
```

Рисунок 2 - Клонирование репозитория

3. Организуйте свой репозиторий в соответствии с моделью ветвления git-flow.

```

Asus@LAPTOP-09A994CG MINGW64 /c/my projects/3
$ cd lw_2.17

Asus@LAPTOP-09A994CG MINGW64 /c/my projects/3/lw_2.17 (main)
$ git checkout -b develop
Switched to a new branch 'develop'

Asus@LAPTOP-09A994CG MINGW64 /c/my projects/3/lw_2.17 (develop)
$

```

Рисунок 3 - Ветвление по модели git-flow

4. Создайте проект PyCharm в папке репозитория.

Этот компьютер > OS (C:) > My projects > 3 > lw\_2.17 >

Имя	Дата изменения	Тип	Размер
Project	29.11.2022 11:49	Папка с файлами	
.gitignore	29.11.2022 11:44	Файл "GITIGNORE"	2 КБ
LICENSE	29.11.2022 11:44	Файл	2 КБ
README.md	29.11.2022 11:44	Файл "MD"	1 КБ

Рисунок 4 - Создание проекта

5. Проработать примеры лабораторной работы.

```

C:\My projects\3\lw_2.17\Project>python primer.py add data.json --name="Сидоров Сидор"
--post="Главный инженер" --year=2012

C:\My projects\3\lw_2.17\Project>primer.py display data.json
+-----+-----+-----+-----+
| No |          Ф.И.О.          |      Должность      |      Год      |
+-----+-----+-----+-----+
|  1 | Сидоров Сидор           |  Главный инженер   |      2012     |
+-----+-----+-----+-----+

C:\My projects\3\lw_2.17\Project>primer.py select data.json --period=10
+-----+-----+-----+-----+
| No |          Ф.И.О.          |      Должность      |      Год      |
+-----+-----+-----+-----+
|  1 | Сидоров Сидор           |  Главный инженер   |      2012     |
+-----+-----+-----+-----+

```

Рисунок 5 – Результат выполнения примера

6. Выполнить индивидуальные задания.

### Задание

Для своего варианта лабораторной работы 2.16 необходимо дополнительно реализовать интерфейс командной строки (CLI).

```
C:\My projects\3\lw_2.17\Project>python ind.py add planes.json --destination="Москва" --num=123 --typ="грузовой"
```

```
C:\My projects\3\lw_2.17\Project>python ind.py add planes.json --destination="Санкт-Петербург" --num=456 --typ="пассажирский"
```

```
C:\My projects\3\lw_2.17\Project>ind.py display planes.json
```

No	Пункт назначения	Номер рейса	Тип самолета
1	Москва	123	грузовой
2	Санкт-Петербург	456	пассажирский

```
C:\My projects\3\lw_2.17\Project>ind.py select planes.json --type="пассажирский"
```

No	Пункт назначения	Номер рейса	Тип самолета
1	Санкт-Петербург	456	пассажирский

Рисунок 6 - Результат выполнения индивидуального задания

### 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`.

```
C:\My projects\3\lw_2.17\Project\Индивидуальные задания>python ind_hard.py planes.json -c add -d Санкт-Петербург -n 456 -t пассажирский
```

```
C:\My projects\3\lw_2.17\Project\Индивидуальные задания>python ind_hard.py planes.json -c display
```

No	Пункт назначения	Номер рейса	Тип самолета
1	Москва	123	грузовой
2	Санкт-Петербург	456	пассажирский
3	Санкт-Петербург	456	пассажирский

```
C:\My projects\3\lw_2.17\Project\Индивидуальные задания>python ind_hard.py planes.json -c select -t грузовой
```

No	Пункт назначения	Номер рейса	Тип самолета
1	Москва	123	грузовой

Рисунок 7 - Результат выполнения задания повышенной сложности

### 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.

## **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()`.

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

Основанный на функции C `getopt`, он позволяет использовать как короткие, так и длинные варианты, включая присвоение значений. На практике для правильной обработки входных данных требуется модуль `sys`. Для этого необходимо заранее загрузить как модуль `sys`, так и модуль `getopt`. Затем из списка входных параметров мы удаляем первый элемент списка (см. код ниже) и сохраняем оставшийся список аргументов командной строки в переменной с именем `arguments_list`.

Аргументы в списке аргументов теперь можно анализировать с помощью метода `getopts()`. Но перед этим нам нужно сообщить `getopts()` о том, какие параметры допустимы.

Для метода `getopt()` необходимо настроить три параметра – список фактических аргументов из `argv`, а также допустимые короткие и длинные параметры.

Сам вызов метода хранится в инструкции `try - catch`, чтобы скрыть ошибки во время оценки. Исключение возникает, если обнаруживается аргумент, который не является частью списка, как определено ранее. Скрипт в Python выведет сообщение об ошибке на экран и выйдет с кодом ошибки 2.

Наконец, аргументы с соответствующими значениями сохраняются в двух переменных с именами `arguments` и `values`. Теперь вы можете легко оценить эти переменные в своем коде. Мы можем использовать цикл `for` для перебора списка распознанных аргументов, одна запись за другой.

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

Начиная с версий Python 2.7 и Python 3.2, в набор стандартных библиотек была включена библиотека `argparse` для обработки аргументов (параметров, ключей) командной строки.

Для начала работы с `argparse` необходимо задать парсер.

Далее, парсеру стоит указать, какие объекты Вы от него ждете.

Если действие (`action`) для данного аргумента не задано, то по умолчанию он будет сохраняться (`store`) в `namespace`, причем мы также можем указать тип этого аргумента (`int`, `boolean` и тд). Если имя возвращаемого аргумента (`dest`) задано, его значение будет сохранено в соответствующем атрибуте `namespace`.

Остановимся на действиях (`actions`). Они могут быть следующими:

`store`: возвращает в пространство имен значение (после необязательного приведения типа). Как уже говорилось, `store` — действие по умолчанию;

`store_const`: в основном используется для флагов. Либо вернет Вам значение, указанное в `const`, либо (если ничего не указано), `None`.

`store_true` / `store_false`: аналог `store_const`, но для булевых `True` и `False`;

`append`: возвращает список путем добавления в него значений аргументов.

`append_const`: возвращение значения, определенного в спецификации аргумента, в список.

`count`: как следует из названия, считает, сколько раз встречается значение данного аргумента.

**Вывод:** были приобретены навыки по работе с данными формата JSON при написании программ с помощью языка программирования Python версии 3.x.