

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»

Выполнила: студентка

2 курса, группы ИВТ-б-о-21-1

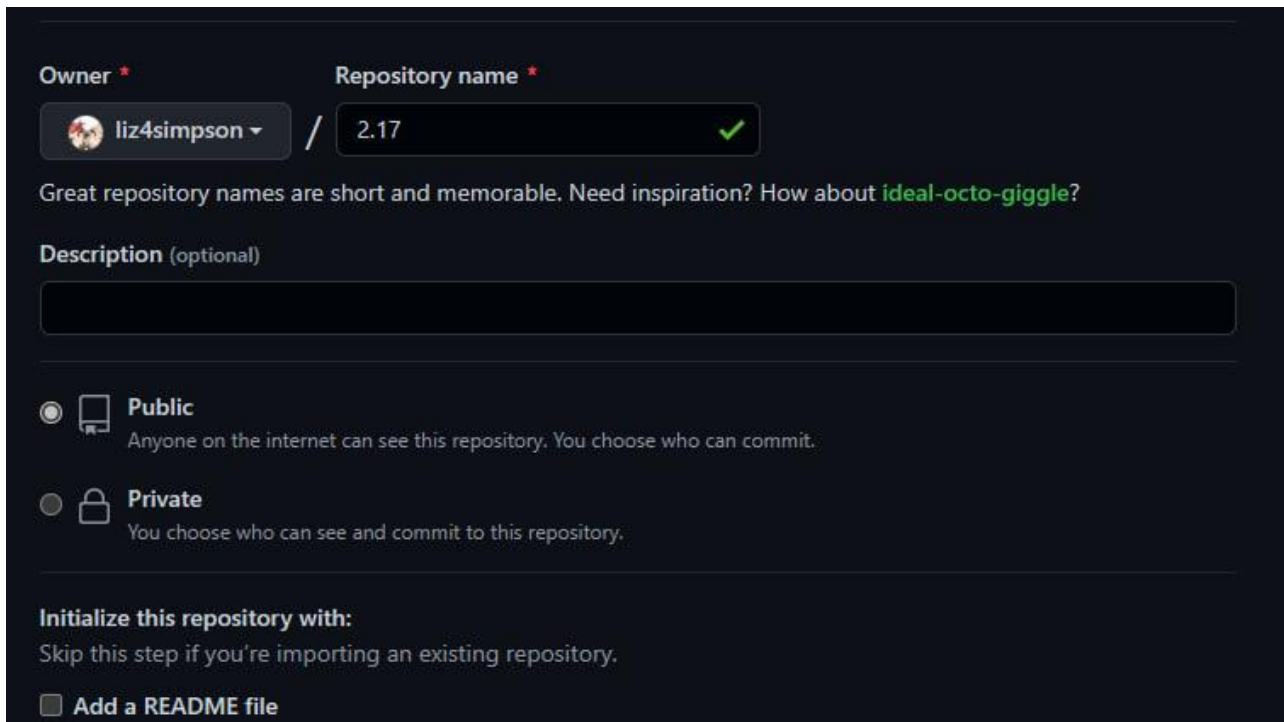
Яковлева Е.А.

Ставрополь 2022

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

**Практическая часть:**

1. Создала репозиторий, клонировала его, изменила файл .gitignore. Организовала репозиторий в соответствии с моделью ветвления git-flow



The screenshot shows the GitHub 'Create new repository' form. At the top, the 'Owner' is set to 'liz4simpson' and the 'Repository name' is '2.17', which is marked as valid with a green checkmark. Below this, a suggestion for repository names is provided: 'Great repository names are short and memorable. Need inspiration? How about **ideal-octo-giggle?**'. The 'Description (optional)' field is empty. Under the 'Visibility' section, the 'Public' option is selected, with the description 'Anyone on the internet can see this repository. You choose who can commit.' The 'Private' option is unselected, with the description 'You choose who can see and commit to this repository.' At the bottom, the 'Initialize this repository with:' section is visible, with the instruction 'Skip this step if you're importing an existing repository.' and a checkbox for 'Add a README file' which is currently unchecked.

Рисунок 1. Создание репозитория

```
# Created by https://www.toptal.com/developers/gitignore/api/python,pycharm
# Edit at https://www.toptal.com/developers/gitignore?templates=python,pycharm

### PyCharm ###
# Covers JetBrains IDEs: IntelliJ, RubyMine, PhpStorm, AppCode, PyCharm, CLion, Android Studio
# Reference: https://intellij-support.jetbrains.com/hc/en-us/articles/206544839
.idea/
.idea
*.json
# User-specific stuff
.idea/**/workspace.xml
.idea/**/tasks.xml
.idea/**/usage.statistics.xml
.idea/**/dictionaries
.idea/**/shelf
/Lab17

# AWS User-specific
.idea/**/aws.xml

# Generated files
.idea/**/contentModel.xml

# Sensitive or high-churn files
.idea/**/dataSources/
```

Рисунок 3. Изменение файла .gitignore

```
Git CMD
C:\Users\Elizaveta>cd /d C:\Users\Elizaveta\Desktop\git\2.17
C:\Users\Elizaveta\Desktop\git\2.17>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/Elizaveta/Desktop/git/2.17/.git/hooks]
C:\Users\Elizaveta\Desktop\git\2.17>
```

Рисунок 4. Организация репозитория в соответствии с git-flow

2. Проработала примеры лабораторной работы.

```
c:\Users\Admin\Desktop\git\Python12-2.17\Primers>python primer1.py display data.json
```

No	Ф.И.О.	Должность	Год
1	Сидоров Сидор	Главный инженер	2012
2	Иванченко Никита	Старший слесарь	2008

```
c:\Users\Admin\Desktop\git\Python12-2.17\Primers>
```

Рисунок 5. Результат работы примера

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

```
PS C:\Users\super\Desktop\Dina\ВУЗ\Программирование на python\Lab2_17\prog> python individual1.py add ind1.json --names 'three' --noes --time '21:21:00'
```

```
PS C:\Users\super\Desktop\Dina\ВУЗ\Программирование на python\Lab2_17\prog> python individual1.py display ind1.json
```

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

Рисунок 6. Результат выполнения индивидуального задания

4. 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.

```
PS C:\Users\super\Desktop\Dina\ВУЗ\Программирование на python\Lab2_17\prog> python individual2.py -c display ind1.json
```

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

Рисунок 7. Результат выполнения индивидуального 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. 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.