

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ  
ФЕДЕРАЦИИ**

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

**Кафедра инфокоммуникаций  
«Установка пакетов в Python. Виртуальные окружения»**

**Отчет по лабораторной работе № 2.16  
по дисциплине «Основы программной инженерии»**

Выполнил студент группы

ПИЖ-б-о-21-1

Трушева В. О. .« » 2023г.

Подпись студента \_\_\_\_\_

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

Проверила Воронкин Р.А. \_\_\_\_\_

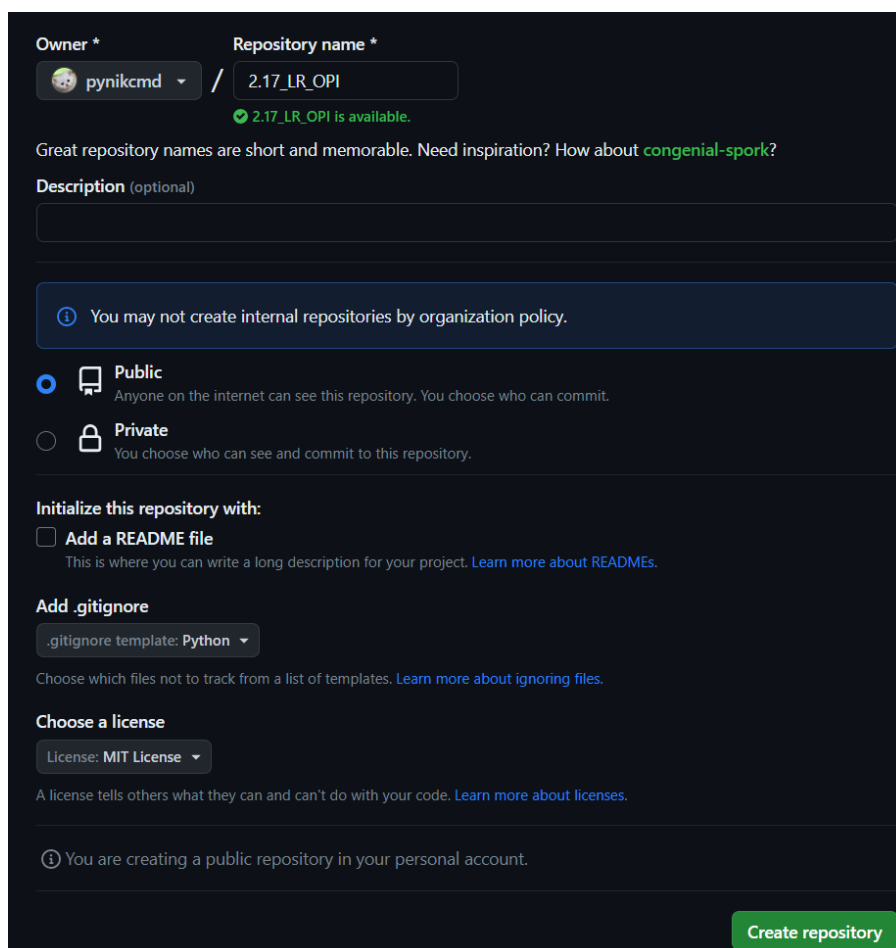
(подпись)

Ставрополь 2023

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

Методика и порядок выполнения работы

1. Изучить теоретический материал работы.
2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python.



The screenshot shows the GitHub 'Create new repository' form. At the top, the 'Owner' is set to 'pynikcmd' and the 'Repository name' is '2.17\_LR\_OPI'. A green checkmark indicates the name is available. Below this, there is a text box for a 'Description' and a warning message: 'You may not create internal repositories by organization policy.' The 'Public' option is selected under 'Visibility'. Under 'Initialize this repository with:', the 'Add a README file' checkbox is unchecked. The '.gitignore' template is set to 'Python'. The 'License' is set to 'MIT License'. At the bottom right, there is a green 'Create repository' button. A final note at the bottom states: 'You are creating a public repository in your personal account.'

Рисунок – Создание репозитория

3. Выполните клонирование созданного репозитория.

```
D:\fgit>git clone https://github.com/pynikcmd/2.17_LR_OPI.git
Cloning into '2.17_LR_OPI'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (4/4), done.
```

Рисунок – Клонирование репозитория

4. Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm.

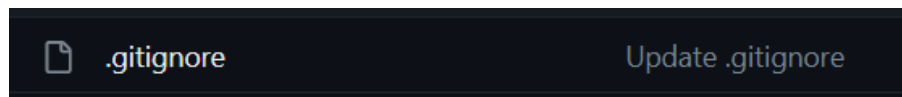


Рисунок – Дополнен .gitignore

5. Организуйте свой репозиторий в соответствие с моделью ветвления git-flow.

```
D:\fgit\2.17_LR_OPI>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? [D:/fgit/2.17_LR_OPI/.git/hooks]
```

Рисунок – Модель git-flow

6. Создайте проект PyCharm в папке репозитория.

7. Проработайте примеры лабораторной работы. Создайте для них отдельные модули языка Python. Зафиксируйте изменения в репозитории.

8. Приведите в отчете скриншоты результатов выполнения примера при различных исходных данных вводимых с клавиатуры.

```
D:\fgit\2.17_LR_OPI\Tasks>python primers.py add primers.json --name="Лавина Яна" --post="Работник" --year=2010
D:\fgit\2.17_LR_OPI\Tasks>python primers.py add primers.json --name="Мируна Валерия" --post="Бухгалтер" --year=2008
D:\fgit\2.17_LR_OPI\Tasks>python primers.py display primers.json
+-----+-----+-----+-----+
| № | Ф.И.О. | Должность | Год |
+-----+-----+-----+-----+
| 1 | Лавина Яна | Работник | 2010 |
+-----+-----+-----+-----+
| 2 | Мируна Валерия | Бухгалтер | 2008 |
+-----+-----+-----+-----+
D:\fgit\2.17_LR_OPI\Tasks>python primers.py select primers.json --period=4
+-----+-----+-----+-----+
| № | Ф.И.О. | Должность | Год |
+-----+-----+-----+-----+
| 1 | Лавина Яна | Работник | 2010 |
+-----+-----+-----+-----+
```

Рисунок – Результат работы программы

9. Зафиксируйте сделанные изменения в репозитории.

10. Приведите в отчете скриншоты работы программ решения индивидуальных заданий. Для своего варианта лабораторной работы 2.16 необходимо дополнительно реализовать интерфейс командной строки (CLI).

```
D:\fgit\2.17_LR_OPI\Tasks>python Ind.py add ind.json --name="Самарина Инна" --number=12332445 --bday=10.09.2010
D:\fgit\2.17_LR_OPI\Tasks>python Ind.py add ind.json --name="Иванов Федор" --number=7487825 --bday=05.09.2020
D:\fgit\2.17_LR_OPI\Tasks>python Ind.py display ind.json
+-----+-----+-----+-----+
| № | Фамилия и имя | Телефон | День рождения |
+-----+-----+-----+-----+
| 1 | Самарина Инна | 12332445 | 2010-09-10 |
+-----+-----+-----+-----+
| 2 | Иванов Федор | 7487825 | 2020-09-05 |
+-----+-----+-----+-----+
```

Рисунок – Результат работы программы

```
D:\fgit\2.17_LR_OPI\Tasks>python Ind.py find ind.json --nomer=7487825
+-----+-----+-----+-----+
| № |          Фамилия и имя          |   Телефон   |   День рождения   |
+-----+-----+-----+-----+
| 1 | Иванов Федор                    | 7487825     | 2020-09-05       |
+-----+-----+-----+-----+
```

Рисунок – Результат работы программы

### 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.

```
D:\fgit\2.17_LR_OPI\Tasks>python ind4.py add ind_click.json --name="Венарина Лиля" --number=2345671234 --bday=09.09.2009
Данные добавлены

D:\fgit\2.17_LR_OPI\Tasks>python ind4.py add ind_click.json --name="Акулин Илья" --number=3459871234 --bday=09.09.2010
Данные добавлены

D:\fgit\2.17_LR_OPI\Tasks>python ind4.py display ind_click.json
+-----+-----+-----+-----+
| № |          Фамилия и имя          |   Телефон   |   День рождения   |
+-----+-----+-----+-----+
| 1 | Венарина Лиля                  | 0           | 2009-09-09       |
+-----+-----+-----+-----+
| 2 | Венарина Лиля                  | 2345671234  | 2009-09-09       |
+-----+-----+-----+-----+
| 3 | Акулин Илья                    | 3459871234  | 2010-09-09       |
+-----+-----+-----+-----+
```

Рисунок – Результат работы программы

```
D:\fgit\2.17_LR_OPI\Tasks>python ind4.py add ind_click.json --name="Акин Ян" --number=12345 --bday=09.09.2010
Данные добавлены

D:\fgit\2.17_LR_OPI\Tasks>python ind4.py add ind_click.json --name="Ливина Яна" --number=12345 --bday=09.09.2010
Данные добавлены

D:\fgit\2.17_LR_OPI\Tasks>python ind4.py find ind_click.json --nomer=12345
+-----+-----+-----+-----+
| № |          Фамилия и имя          |   Телефон   |   День рождения   |
+-----+-----+-----+-----+
| 1 | Акин Ян                        | 12345       | 2010-09-09       |
+-----+-----+-----+-----+
| 2 | Ливина Яна                     | 12345       | 2010-09-09       |
+-----+-----+-----+-----+
```

Рисунок – Результат работы программы

11. Зафиксируйте сделанные изменения в репозитории.

12. Добавьте отчет по лабораторной работе в формате PDF в папку doc репозитория. Зафиксируйте изменения.

13. Выполните слияние ветки для разработки с веткой master/main.

14. Отправьте сделанные изменения на сервер GitHub.

15. Отправьте адрес репозитория GitHub на электронный адрес преподавателя.

#### Контрольные вопросы

1. В чем отличие терминала и консоли?

Терминал – устройство или ПО, выступающее посредником между человеком и вычислительной системой. Терминал - это среда ввода и вывода текста. Консоль - это физический терминал; приборная панель, содержащая элементы управления компьютером. Консоль - это тип терминала.

2. Что такое консольное приложение?

Консольное приложение — это программа, которая работает в командной строке (консоли) операционной системы. В отличие от графических приложений, которые имеют графический пользовательский

интерфейс, консольные приложения взаимодействуют с пользователем через командную строку и текстовый 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`?

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. 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`.

```
# Include standard modules import getopt, sys

# Get full command-line arguments full_cmd_arguments = sys.argv

# Keep all but the first

argument_list = full_cmd_arguments[1:] print(argument_list)
```

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

```
short_options = "ho:v"

long_options = ["help", "output=", "verbose"]
```



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.

```
try:

    arguments, values = getopt.getopt(argument_list, short_options,
long_options)

    except getopt.error as err:

        # Output error, and return with an error code print(str(err))

        sys.exit(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.

```
# Evaluate given options

for current_argument, current_value in arguments:

    if current_argument in ("-v", "--verbose"):

        print("Enabling verbose mode")

    elif current_argument in ("-h", "--help"):

        print("Displaying help")

    elif current_argument in ("-o", "--output"):
```

```
print(f'Enabling special output mode ({current_value})')
```

Ниже вы можете увидеть результат выполнения этого кода. Далее показано, как программа реагирует как на допустимые, так и на недопустимые программные аргументы:

```
$ python arguments-getopt.py -h Displaying help
```

```
$ python arguments-getopt.py --help Displaying help
```

```
$ python arguments-getopt.py --output=green --help -v Enabling special output mode (green)
```

```
Displaying help Enabling verbose mode
```

```
$ python arguments-getopt.py -verbose option -e not recognized
```

Последний вызов нашей программы поначалу может показаться немного запутанным. Чтобы понять это, вам нужно знать, что сокращенные параметры (иногда также называемые флагами) могут использоваться вместе с одним тире. Это позволяет вашему инструменту легче воспринимать множество вариантов.

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 `ср`, имеющая минимум 2 таких аргумента («`ср 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.