

**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 № 12**

Дисциплина: «Программирование на Python»

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

Выполнил: студент 2 курса

группы ИВТ-б-о-21-1

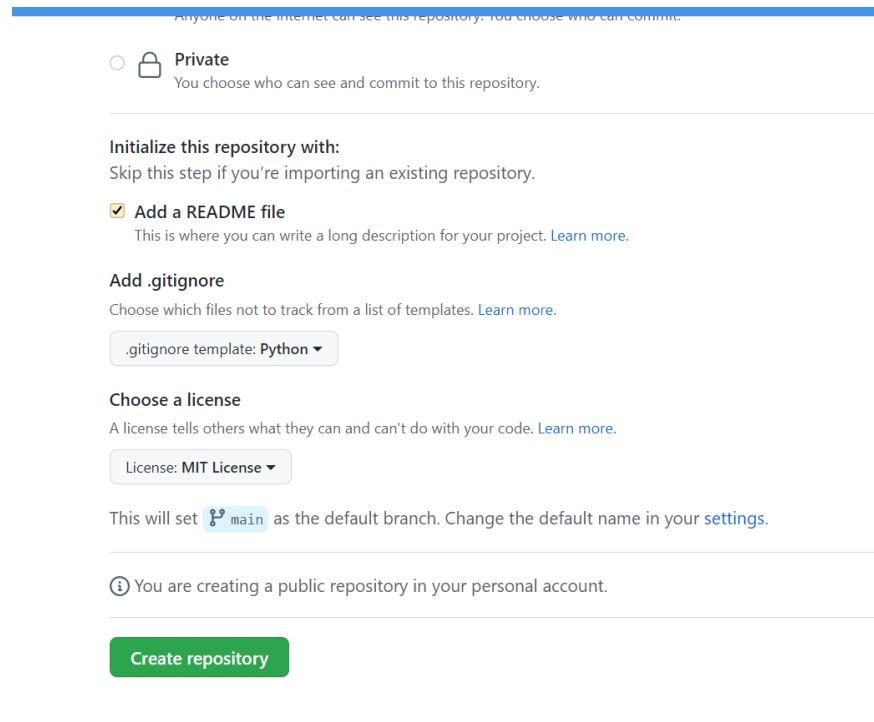
Уланбекова Айканыш Уланбековна

Ставрополь 2022

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

**Порядок выполнения работы:**

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



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](#).

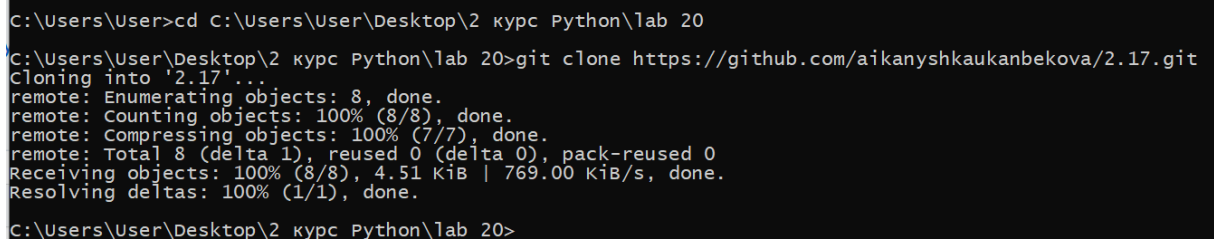
---

You are creating a public repository in your personal account.

Create repository

Рисунок 1. Создание репозитория

2. Выполните клонирование созданного репозитория.



```
c:\Users\User>cd c:\Users\User\Desktop\2 кypc Python\lab 20
c:\Users\User\Desktop\2 кypc Python\lab 20>git clone https://github.com/aikanyshkaukanbekova/2.17.git
Cloning into '2.17'...
remote: Enumerating objects: 8, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 8 (delta 1), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (8/8), 4.51 KiB | 769.00 KiB/s, done.
Resolving deltas: 100% (1/1), done.
c:\Users\User\Desktop\2 кypc Python\lab 20>
```

Рисунок 2. Клонирование репозитория

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

```

C:\Users\User\Desktop\2 кypc Python\lab 10\lab-10>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/User/Desktop/2 кypc Python/lab 10/lab-10/.git/hooks]
C:\Users\User\Desktop\2 кypc Python\lab 10\lab-10>

```

Рисунок 4. Организован модель ветвления git flow

5.Проработайте примеры лабораторной работы. Создайте для них отдельный модуль языка Python. Зафиксируйте изменения в репозитории.

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import argparse
import json
import os.path
from datetime import date

def add_worker(staff, name, post, year):
    """
    Добавить данные о работнике.
    """
    staff.append(
        {
            "name": name,
            "post": post,
            "year": year
        }
    )
    return staff

def display_workers(staff):
    """
    Отобразить список работников.
    """
    # Проверить, что список работников не пуст.
    if staff:
        # Заголовок таблицы.
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 20,
            '-' * 8
        )
        print(line)
        print(
            '| {:^4} | {:^30} | {:^20} | {:^8} |'.format(
                "No",
                "Ф.И.О.",
                "Должность",
                "Год"
            )
        )

```

```

    )
    )
    print(line)

    # Вывести данные о всех сотрудниках.
    for idx, worker in enumerate(staff, 1):
        print(
            '| {:>4} | {:<30} | {:<20} | {:>8} |'.format(
                idx,
                worker.get('name', ''),
                worker.get('post', ''),
                worker.get('year', 0)
            )
        )
    print(line)

else:
    print("Список работников пуст.")

def select_workers(staff, period):
    """
    Выбрать работников с заданным стажем.
    """
    # Получить текущую дату.
    today = date.today()

    # Сформировать список работников.
    result = []
    for employee in staff:
        if today.year - employee.get('year', today.year) >= period:
            result.append(employee)

    # Возвратить список выбранных работников.
    return result

def save_workers(file_name, staff):
    """
    Сохранить всех работников в файл JSON.
    """
    # Открыть файл с заданным именем для записи.
    with open(file_name, "w", encoding="utf-8", errors="ignore") as fout:
        # Выполнить сериализацию данных в формат JSON.
        # Для поддержки кириллицы установим ensure_ascii=False
        json.dump(staff, fout, ensure_ascii=False, indent=4)

def load_workers(file_name):
    """
    Загрузить всех работников из файла JSON.
    """
    # Открыть файл с заданным именем для чтения.
    with open(file_name, "r", encoding="utf-8", errors="ignore") as fin:
        return json.load(fin)

def main(command_line=None):
    # Создать родительский парсер для определения имени файла.
    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument(
        "filename",
        action="store",
        help="The data file name"
    )

```

```

)

# Создать основной парсер командной строки.
parser = argparse.ArgumentParser("workers")
parser.add_argument(
    "--version",
    action="version",
    version="% (prog)s 0.1.0"
)

subparsers = parser.add_subparsers(dest="command")

# Создать субпарсер для добавления работника.
add = subparsers.add_parser(
    "add",
    parents=[file_parser],
    help="Add a new worker"
)
add.add_argument(
    "-n",
    "--name",
    action="store",
    required=True,
    help="The worker's name"
)
add.add_argument(
    "-p",
    "--post",
    action="store",
    help="The worker's post"
)
add.add_argument(
    "-y",
    "--year",
    action="store",
    type=int,
    required=True,
    help="The year of hiring"
)

# Создать субпарсер для отображения всех работников.
_ = subparsers.add_parser(
    "display",
    parents=[file_parser],
    help="Display all workers"
)

# Создать субпарсер для выбора работников.
select = subparsers.add_parser(
    "select",
    parents=[file_parser],
    help="Select the workers"
)
select.add_argument(
    "-p",
    "--period",
    action="store",
    type=int,
    required=True,
    help="The required period"
)

# Выполнить разбор аргументов командной строки.
args = parser.parse_args(command_line)

```

```

# Загрузить всех работников из файла, если файл существует .
is_dirty = False
if os.path.exists(args.filename):
    workers = load_workers(args.filename)
else:
    workers = []

# Добавить работника.
if args.command == "add":
    workers = add_worker(
        workers,
        args.name,
        args.post,
        args.year
    )
    is_dirty = True

# Отобразить всех работников.
elif args.command == "display":
    display_workers(workers)

# Выбрать требуемых работников.
elif args.command == "select":
    selected = select_workers(workers, args.period)
    display_workers(selected)

# Сохранить данные в файл, если список работников был изменен.
if is_dirty:
    save_workers(args.filename, workers)

if __name__ == '__main__':
    main()

```

Рисунок 5. Примеры лаб работы

## 6. Индивидуальное задание

### Вариант 9.

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

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import argparse
import json
import os.path

def add_route(routes, start, finish, number):
    """
    Добавить данные о маршруте
    """
    routes.append(

```

```

        {
            'start': start,
            'finish': finish,
            'number': number
        }
    )
    return routes

def display_route(routes):
    """
    Отобразить список маршрутов
    """
    if routes:
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 20,
            '-' * 8
        )
        print(line)
        print(
            '| {:^4} | {:^30} | {:^20} | {:^8} |'.format(
                "№",
                "Начальный пункт",
                "Конечный пункт",
                "Номер маршрута"
            )
        )
        print(line)

        for idx, worker in enumerate(routes, 1):
            print(
                '| {:>4} | {:<30} | {:<20} | {:>8} |'.format(
                    idx,
                    worker.get('start', ''),
                    worker.get('finish', ''),
                    worker.get('number', 0)
                )
            )
        print(line)
    else:
        print("Список маршрутов пуст.")

def select_route(routes, period):
    """
    Выбрать маршрут
    """
    result = []
    for employee in routes:
        if employee.get('number') == period:
            result.append(employee)

    return result

def save_routes(file_name, routes):
    """
    Сохранить всех работников в файл JSON.
    """
    with open(file_name, "w", encoding="utf-8") as fout:
        json.dump(routes, fout, ensure_ascii=False, indent=4)

```

```

def load_routes(file_name):
    """
    Загрузить всех работников из файла JSON.
    """
    with open(file_name, "r", encoding="utf-8") as fin:
        return json.load(fin)

def main(command_line=None):
    # Создать родительский парсер для определения имени файла.
    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument(
        "filename",
        action="store",
        help="The data file name"
    )
    # Создать основной парсер командной строки.
    parser = argparse.ArgumentParser("routes")
    parser.add_argument(
        "--version",
        action="version",
        version="% (prog)s 0.1.0"
    )
    subparsers = parser.add_subparsers(dest="command")
    # Создать субпарсер для добавления маршрута.
    add = subparsers.add_parser(
        "add",
        parents=[file_parser],
        help="Add a new route"
    )
    add.add_argument(
        "-s",
        "--start",
        action="store",
        required=True,
        help="The start of the route"
    )
    add.add_argument(
        "-f",
        "--finish",
        action="store",
        help="The finish of the route"
    )
    add.add_argument(
        "-n",
        "--number",
        action="store",
        type=int,
        required=True,
        help="The number of the route"
    )
    # Создать субпарсер для отображения всех маршрутов.
    _ = subparsers.add_parser(
        "display",
        parents=[file_parser],
        help="Display all routes"
    )
    # Создать субпарсер для выбора маршрута.
    select = subparsers.add_parser(
        "select",
        parents=[file_parser],
        help="Select the route"
    )

```



```

select.add_argument(
    "-N",
    "--numb",
    action="store",
    type=int,
    required=True,
    help="The route"
)
# Выполнить разбор аргументов командной строки.
args = parser.parse_args(command_line)
# Загрузить все маршруты из файла, если файл существует.
is_dirty = False
if os.path.exists(args.filename):
    routes = load_routes(args.filename)
else:
    routes = []
# Добавить маршрут.
if args.command == "add":
    routes = add_route(
        routes,
        args.start,
        args.finish,
        args.number
    )
    is_dirty = True
# Отобразить все маршруты.
elif args.command == "display":
    display_route(routes)
# Выбрать требуемые маршруты.
elif args.command == "select":
    selected = select_route(routes, args.numb)
    display_route(selected)
# Сохранить данные в файл, если список маршрутов был изменен.
if is_dirty:
    save_routes(args.filename, routes)

if __name__ == '__main__':
    main()

```

Рисунки 6. Выполненное индивидуальное задание

### 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`.

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import json
import click

@click.group()
def cl():
    pass

```

```

@cl.command()
@click.argument("routes")
@click.option("-s", "--start")
@click.option("-f", "--finish")
@click.option("-n", "--number")
def add_route(routes, start, finish, number):
    """
    Добавить данные о маршруте
    """
    routes = load_routes(routes)
    routes.append(
        {
            'start': start,
            'finish': finish,
            'number': number
        }
    )
    with open(routes, "w", encoding="utf-8") as file_out:
        json.dump(routes, file_out, ensure_ascii=False, indent=4)
    click.secho("Маршрут добавлен", fg='orange')

@cl.command()
@click.argument('routes')
def display_route(routes):
    """
    Отобразить список маршрутов
    """
    if routes:
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 20,
            '-' * 8
        )
        print(line)
        print(
            '| {:^4} | {:^30} | {:^20} | {:^8} |'.format(
                "№",
                "Начальный пункт",
                "Конечный пункт",
                "Номер маршрута"
            )
        )
        print(line)

        for idx, worker in enumerate(routes, 1):
            print(
                '| {:>4} | {:<30} | {:<20} | {:>8} |'.format(
                    idx,
                    worker.get('start', ''),
                    worker.get('finish', ''),
                    worker.get('number', 0)
                )
            )
            print(line)
    else:
        print("Список маршрутов пуст.")

@cl.command()
@click.argument('routes')
@click.option("-N", "--numb")
def select_route(routes, period):

```

```

"""
Выбрать маршрут
"""
result = []
for employee in routes:
    if employee.get('number') == period:
        result.append(employee)

return result

def load_routes(routes):
    with open(routes, "r", encoding="utf-8") as f_in:
        return json.load(f_in)

def main():
    cl()

if __name__ == '__main__':
    main()

```

Рисунок 8. Выполненное индивидуальное задание

8. Сделала коммит, выполнил слияние с веткой main, и запустил изменения в уд. репозиторий.

```

C:\Users\User\Desktop\2 кypc Python\lab 20>cd C:\Users\User\Desktop\2 кypc Python\lab 20\2.17
C:\Users\User\Desktop\2 кypc Python\lab 20\2.17>git add .
C:\Users\User\Desktop\2 кypc Python\lab 20\2.17>git commit -m "plus"
[main 9940ecd] plus
3 files changed, 485 insertions(+)
create mode 100644 indiv 1.py
create mode 100644 level_up.py
create mode 100644 primer 1.py
C:\Users\User\Desktop\2 кypc Python\lab 20\2.17>git push
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 8 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 3.84 KiB | 1.28 MiB/s, done.
Total 5 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/aikanyshkaukanbekova/2.17.git
   1a7e35a..9940ecd  main -> main

```

Рисунок 8. Сохранения

## 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.