МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Кафедра инфокоммуникаций

Основы кросс платформенного программирования

Отчет по лабораторной работе№2.17

Разработка приложений с интерфейсом командной строки (CLI) в Python3

Выполнил студент группы
ИТС-б-о-21-1 (2)
Якупов Э.А« »_____20__г.
Подпись студента_____
Работа защищена« »_____20__г.
Проверил к.т.н., доцент
Кафедры инфокоммуникаций
Воронкин Р.А.

_____
(подпись)

Ставрополь 2022

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

Порядок выполнения работы:

1) Создадим общедоступный репозиторий на GitHub (https://github.com/Blekroyt/Fox7.git)

2) Решим задачи с помощью языка программирования Python3. И отправим их на GitHub.

Ход работы:

Пример1

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import argparse
import json
import os.path
import sys
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

```

Рис.1 примера 1

```python
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)
```

Рис.2 примера 1

```python
            )
        )
        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.
    """
```

Рис.3 примера 1

```
    # 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") as fout:
        # Выполнить сериализацию данных в формат JSON.
        # Для поддержки кирилицы установим ensure_ascii=False
        json.dump(staff, fout, ensure_ascii=False, indent=4)

```

Рис.4 примера 1

```
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") 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",
```

Рис.5 примера 1

```
114         action="version",
115         version="%(prog)s 0.1.0"
116     )
117     subparsers = parser.add_subparsers(dest="command")
118
119     # Создать субпарсер для добавления работника.
120     add = subparsers.add_parser(
121         "add",
122         parents=[file_parser],
123         help="Add a new worker"
124     )
125     add.add_argument(
126         "-n",
127         "--name",
128         action="store",
129         required=True,
130         help="The worker's name"
131     )
132     add.add_argument(
133         "-p",
134         "--post",
135         action="store",
136         help="The worker's post"
```

Рис.6 примера 1

```
137     )
138     add.add_argument(
139         "-y",
140         "--year",
141         action="store",
142         type=int,
143         required=True,
144         help="The year of hiring"
145     )
146     # Создать субпарсер для отображения всех работников.
147     _ = subparsers.add_parser(
148         "display",
149         parents=[file_parser],
150         help="Display all workers"
151     )
152     # Создать субпарсер для выбора работников.
153     select = subparsers.add_parser(
154         "select",
155         parents=[file_parser],
156         help="Select the workers"
157     )
158     select.add_argument(
159         "-P",
```

Рис.7 примера 1

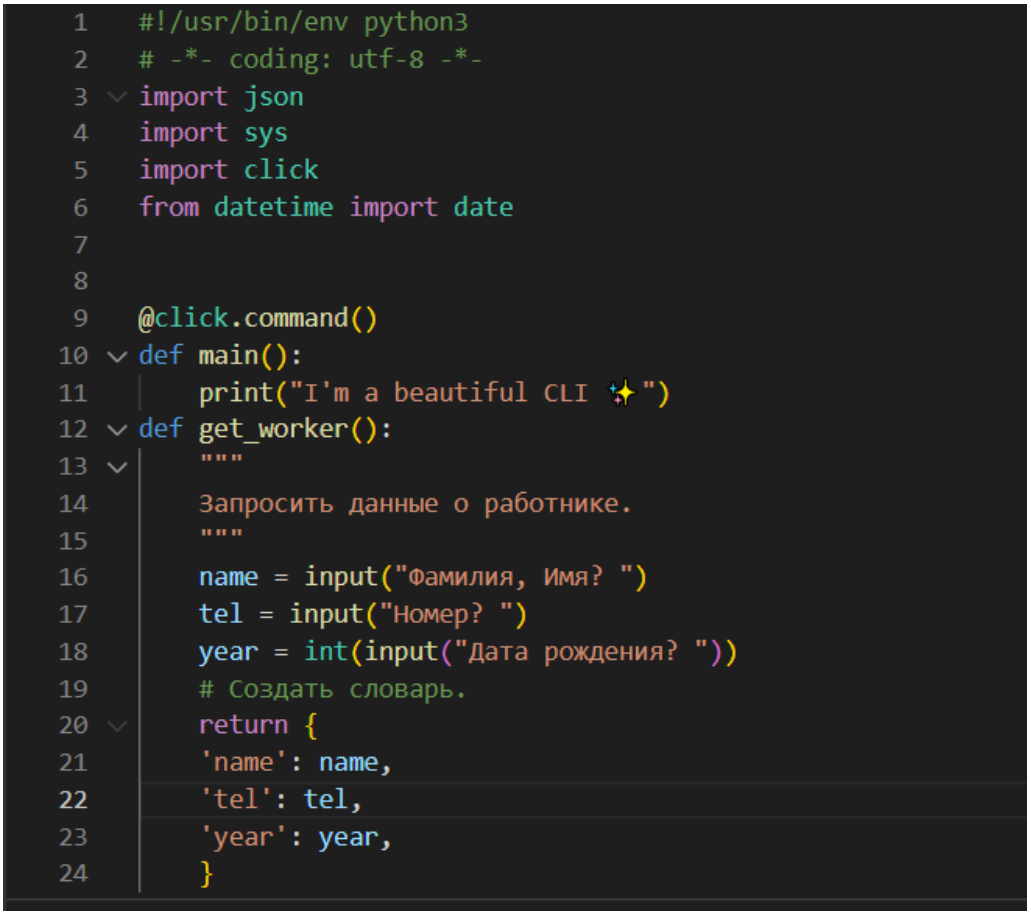
```
160         "--period",
161         action="store",
162         type=int,
163         required=True,
164         help="The required period"
165     )
166     # Выполнить разбор аргументов командной строки.
167     args = parser.parse_args(command_line)
168 
169     # Загрузить всех работников из файла, если файл существует.
170     is_dirty = False
171     if os.path.exists(args.filename):
172         workers = load_workers(args.filename)
173     else:
174         workers = []
175 
176     # Добавить работника.
177     if args.command == "add":
178         workers = add_worker(
179             workers,
180             args.name,
181             args.post,
182             args.year
183         )
```

Рис.8 примера 1

```
184         is_dirty = True
185 
186     # Отобразить всех работников.
187     elif args.command == "display":
188         display_workers(workers)
189 
190     # Выбрать требуемых рааботников.
191     elif args.command == "select":
192         selected = select_workers(workers, args.period)
193         display_workers(selected)
194 
195     # Сохранить данные в файл, если список работников был изменен.
196     if is_dirty:
197         save_workers(args.filename, workers)
198 
199 
200 if __name__ == '__main__':
201     main()
```

Рис.9 примера 1

Индивидуальная



```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import json
import sys
import click
from datetime import date


@click.command()
def main():
    print("I'm a beautiful CLI ✨")
def get_worker():
    """
    Запросить данные о работнике.
    """
    name = input("Фамилия, Имя? ")
    tel = input("Номер? ")
    year = int(input("Дата рождения? "))
    # Создать словарь.
    return {
    'name': name,
    'tel': tel,
    'year': year,
    }
```

Рис.10 индивидуальная работа

1. В чем отличие терминала и консоли?

Консоль - это совокупность устройств, которые позволяет вам взаимодействовать с устройством. Раньше консолью мог быть обычный принтер (в принципе и сейчас может), тогда это будет текстовая консоль/терминал. Не надо путать с текстовым интерфейсом. Терминал - По сути это так же устройство для ввода и вывода информации но>. Консоль, это уже обёртка над терминалом.

2. Что такое консольное приложение?

Консольным приложением называется программа, которая не имеет графического интерфейса - окон, и которая работает в текстовом режиме в черно-белой консоли

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

Click — это Python-пакет для создания красивых интерфейсов командной строки с минимальным количеством кода и

возможностью компоновки. Это один из лучших Python- пакетов для создания CLI, и с ним очень удобно работать.

4. Какие особенности построение CLI с использованием модуля sys ?

Модуль sys реализует аргументы командной строки в простой структуре списка с именем sys.argv На практике для правильной обработки входных данных требуется модуль sys . Для этого необходимо предварительно загрузить как модуль sys , так и модуль getopt . ... Генерация интерфейсов командной строки (CLI) с помощью Fire в Python. Модуль Shutil в Python.

5. Какие особенности построение CLI с использованием модуля getopt ?

чтобы упростить написание кода, придерживающегося стандартных соглашений. Функция GNU getopt_long(), является совместимой с getopt(), а также упрощает разбор длинных опций. getopt. Объявление: #include <unistd.h> int getopt(int argc, char *argv[], const char *optstring);

6. Какие особенности построение CLI с использованием модуля argparse?

Использование модуля argparse в Python для создания интерфейса командной строки, обработки позиционных и необязательных аргументов, их комбинирование с подробными примерами..... Интерфейс командной строки в Python также известен как CLI,

Вывод: приобрели навыки построения приложений с интерфейсом командной строки с помощью языка программирования Python версии 3.x.