

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №4
дисциплины «Программирование на языке Python»

Вариант_15_

Выполнила:
Маньшина Дарья Алексеевна
2 курс, группа ИТС-б-о-22-1,
11.03.02 «Инфокоммуникационные
технологии и системы связи»,
направленность (профиль)
«Инфокоммуникационные системы и
сети», очная форма обучения

(подпись)

Руководитель практики:
Воронкин Р. А., канд. тех. наук,
доцент кафедры
инфокоммуникаций

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

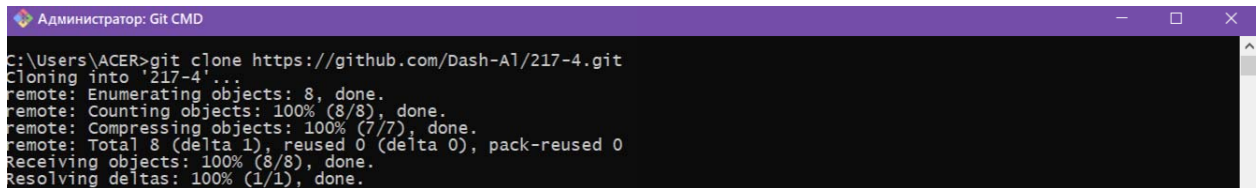
Ставрополь, 2023 г.

Тема: разработка приложений с интерфейсом командной строки (CLI) в Python3

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

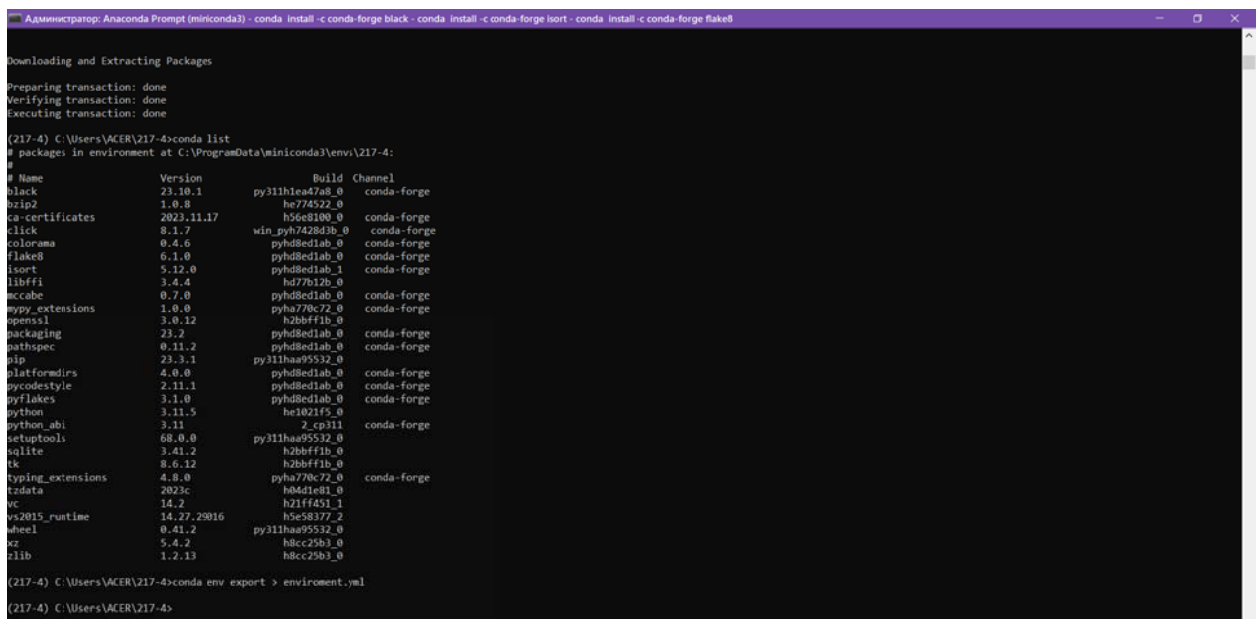
Ход работы:

1. Создадим новый репозиторий. Клонировем его и сделаем способ ветвления git-flow. Также установим пакеты: black, isort, flake8



```
Администратор: Git CMD
C:\Users\ACER>git clone https://github.com/Dash-A1/217-4.git
Cloning into '217-4'...
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), done.
Resolving deltas: 100% (1/1), done.
```

Рисунок 1. Подготовка к лабораторной работе



```
Администратор: Anaconda Prompt (miniconda3) - conda install -c conda-forge black -c conda-forge isort -c conda-forge flake8

Downloading and Extracting Packages
Preparing transaction: done
Verifying transaction: done
Executing transaction: done

(217-4) C:\Users\ACER\217-4>conda list
# packages in environment at C:\ProgramData\miniconda3\envs\217-4:
#
# Name                    Version            Build                Channel
black                     23.10.1            py311h1ae72a9_0      conda-forge
biip2                     1.0.9              h2742522_0           conda-forge
ca-certificates           2023.11.17         h56e8100_0           conda-forge
click                     8.1.7              win_pyh7428d3b_0     conda-forge
colorama                  0.4.6              pyhd8ed1ab_0         conda-forge
flake8                    6.1.0              pyhd8ed1ab_0         conda-forge
isort                      5.12.0             pyhd8ed1ab_1         conda-forge
libffi                     3.4.4              hd77b12b_0           conda-forge
libcabe                   0.7.0              pyhd8ed1ab_0         conda-forge
mypy_extensions           1.0.0              pyha770c72_0         conda-forge
openssl                   3.0.12             h2b6ff1b_0           conda-forge
packaging                  23.2               pyhd8ed1ab_0         conda-forge
pathspec                   0.11.2             pyhd8ed1ab_0         conda-forge
pip                       23.3.1             py311haa95532_0      conda-forge
platformdirs               4.0.0              pyhd8ed1ab_0         conda-forge
pycodestyle                2.11.1             pyhd8ed1ab_0         conda-forge
pyflakes                   3.1.0              pyhd8ed1ab_0         conda-forge
python                     3.11.5             hc1021f5_0           conda-forge
python_abi                 3.11               2_cp311              conda-forge
setuptools                 68.0.0             py311haa95532_0      conda-forge
sqlite                     3.41.2             h2b6ff1b_0           conda-forge
tk                          8.6.12             h2b6ff1b_0           conda-forge
typing_extensions          4.8.0              pyha770c72_0         conda-forge
tzdata                     2023c              h04d1e81_0           conda-forge
vc                          14.2               h21ffa51_1           conda-forge
vs2015_runtime             14.27.29016        h5e58377_2           conda-forge
wheel                      0.41.2             py311haa95532_0      conda-forge
xz                          5.4.2              h8cc25b3_0           conda-forge
zlib                       1.2.13             h8cc25b3_0           conda-forge

(217-4) C:\Users\ACER\217-4>conda env export > environment.yml
(217-4) C:\Users\ACER\217-4>
```

Рисунок 2. Установка пакетов

2. Проработка примеров лабораторной работы.

Пример 1.

Программа:

```
#!/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):
    staff.append(
        {
            "name": name,
            "post": post,
            "year": year
        }
    )
    return staff
def display_workers(staff):
    # Проверить, что список работников не пуст.
    if staff:
        # Заголовок таблицы.
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 20,
            '-' * 8
        )
        print(line)
        print(
            '| {:^4} | {:^30} | {:^20} | {:^8} |'.format(
                "№",
                "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):
    # 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):
    # 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 кириллицы установим ensure_ascii=False
        json.dump(staff, fout, ensure_ascii=False, indent=4)

def load_workers(file_name):
    # 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",
        action="version",
        version="%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()

```

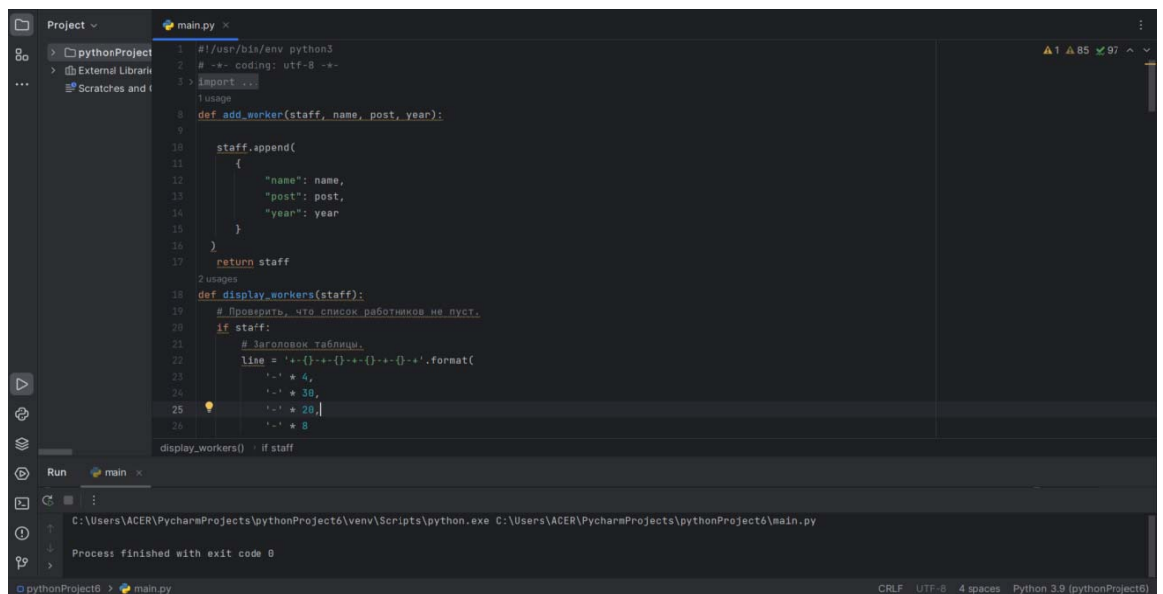


Рисунок 3. Программа примера №1

3. Выполнение индивидуального задания, согласно указанному варианту.

Условие: использовать словарь, содержащий следующие ключи: фамилия, имя; знак Зодиака; дата рождения (список из трёх чисел). Написать программу, выполняющую следующие действия: ввод с клавиатуры данных в список, состоящий из словарей заданной структуры; записи должны быть упорядочены по датам рождения; вывод на экран информацию о людях, родившихся под знаком, название которого введено с клавиатуры; если таких нет, выдать на дисплей соответствующее сообщение. Оформив каждую команду в виде отдельной функции. Дополнительно реализовать сохранение и чтение данных из файла формата JSON. Дополнительно реализовать интерфейс командной строки (CLI).

Программа:

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

def main():
    # Создание парсера аргументов командной строки
    parser = argparse.ArgumentParser()
    parser.add_argument("filename", help="Имя файла для сохранения/загрузки данных")
    args = parser.parse_args()

    def add_person(people):
        last_name = input("Введите фамилию: ")
        first_name = input("Введите имя: ")
        zodiac = input("Введите знак Зодиака: ")
        birth_date = [int(x) for x in input("Введите дату рождения (через пробел): ").split()]
        person = {
            "фамилия": last_name,
            "имя": first_name,
            "знак Зодиака": zodiac,
            "дата рождения": birth_date
        }
        people.append(person)
        people.sort(key=lambda x: x["дата рождения"])
    def search_by_zodiac(people):
        zodiac = input("Введите знак Зодиака для поиска: ")
        found = False
        for person in people:
            if person["знак Зодиака"] == zodiac:
                print("Фамилия:", person["фамилия"])
                print("Имя:", person["имя"])
                print("Знак Зодиака:", person["знак Зодиака"])
                print("Дата рождения:", " ".join(str(x) for x in person["дата рождения"]))
                print()
```

```

found = True
if not found:
    print("Люди с указанным знаком Зодиака не найдены.")
def save_to_file(filename, data):
    with open(filename, "w") as file:
        json.dump(data, file)
def load_from_file(filename):
    with open(filename, "r") as file:
        data = json.load(file)
    return data
def main():
    people = []
    filename = "data.json"
    while True:
        print("1. Добавить человека")
        print("2. Поиск по знаку Зодиака")
        print("3. Сохранить данные в файл")
        print("4. Загрузить данные из файла")
        print("5. Выйти")
        choice = input("Выберите действие: ")
        if choice == "1":
            add_person(people)
        elif choice == "2":
            search_by_zodiac(people)
        elif choice == "3":
            save_to_file(filename, people)
        print("Данные сохранены в файл:", filename)
        elif choice == "4":
            people = load_from_file(filename)
        print("Данные загружены из файла:", filename)
        elif choice == "5":
            break
        else:
            print("Некорректный выбор.")
if __name__ == "__main__":
    main()

```

Результат:

The screenshot shows a code editor with a Python file named `main.py`. The code implements a menu-driven application for managing a list of people. The menu options are: 1. Add person, 2. Search by zodiac sign, 3. Save data to file, 4. Load data from file, and 5. Exit. The code uses a list `people` to store person data, which is a dictionary containing `last_name`, `first_name`, `zodiac`, and `birth_date`. The `search_by_zodiac` function searches for people with a specific zodiac sign and prints their details. The execution output at the bottom shows the user selecting option 2, entering 'Козерог' as the zodiac sign, and displaying the details of a person named Ivan Ivanovich, born on 1/1/2001.

```

pythonProject8  Version control  main  main.py
pythonProject8  18  birth_date = [int(x) for x in input("Введите дату рождения (через пробел): ").split()]
                  19  person = {
                  20      "фамилия": last_name,
                  21      "имя": first_name,
                  22      "знак Зодиака": zodiac,
                  23      "дата рождения": birth_date
                  24  }
                  25  people.append(person)
                  26  people.sort(key=lambda x: x["дата рождения"])
                  27  def search_by_zodiac(people):
                  28      zodiac = input("Введите знак Зодиака для поиска: ")
                  29      found = False
                  30      for person in people:
                  31          if person["знак Зодиака"] == zodiac:
                  32              print("Фамилия:", person["фамилия"])
                  33              print("Имя:", person["имя"])
                  34              print("Знак Зодиака:", person["знак Зодиака"])
                  35              print("Дата рождения:", "/".join(str(x) for x in person["дата рождения"]))
                  36  search_by_zodiac()

main  1. Добавить человека
      2. Поиск по знаку Зодиака
      3. Сохранить данные в файл

```

Рисунок 4. Результат работы индивидуального задания

Вывод: в ходе лабораторной работы приобрела навыки построения приложений с интерфейсом командной строки с помощью языка программирования Python версии 3.x.

Контрольные вопросы

1. В чем отличие терминала и консоли?

Терминал - это программа, которая позволяет пользователю взаимодействовать с операционной системой. Она обычно предоставляет интерфейс командной строки, где пользователь может вводить команды и получать результаты.

Консоль - это устройство или программа, предназначенная для взаимодействия с пользователем. В контексте компьютерной техники, консоль обычно означает терминал, который подключен к компьютеру.

Таким образом, терминал и консоль - это два термина, которые часто используются взаимозаменяемо, но они имеют разные значения в других контекстах.

2. Что такое консольное приложение?

Консольное приложение - это компьютерная программа, которая работает в текстовом режиме, то есть взаимодействует с пользователем через текстовый интерфейс. Консольные приложения обычно используются для выполнения простых задач или для отладки других программ.

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

Python предоставляет несколько модулей для создания приложений командной строки: sys, argparse, getopt.

4. Какие особенности построение CLI с использованием модуля sys?

Модуль sys позволяет обрабатывать аргументы командной строки и выводить информацию в консоль.

5. Какие особенности построение CLI с использованием модуля getopt?

Модуль getopt позволяет обрабатывать опции командной строки, используя формат, аналогичный опции GNU.

6. Какие особенности построение CLI с использованием модуля argparse?

Модуль argparse позволяет создавать мощные и гибкие интерфейсы командной строки. Он поддерживает различные типы аргументов (позиционные, именованные и т. д.), а также позволяет задавать ограничения на аргументы и значения по умолчанию.