

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

Институт цифрового развития
Кафедра инфокоммуникаций

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

ОТЧЕТ
по лабораторной работе №19_2
дисциплины
«Основы программной инженерии»

Выполнила:

Кувшин Ирина Анатольевна
2 курс, группа ПИЖ-б-о-21-1,
011.03.04 «Программная инженерия»,
направленность (профиль) «Разработка
и сопровождение программного
обеспечения», очная форма обучения

(подпись)

Проверил:

(подпись)

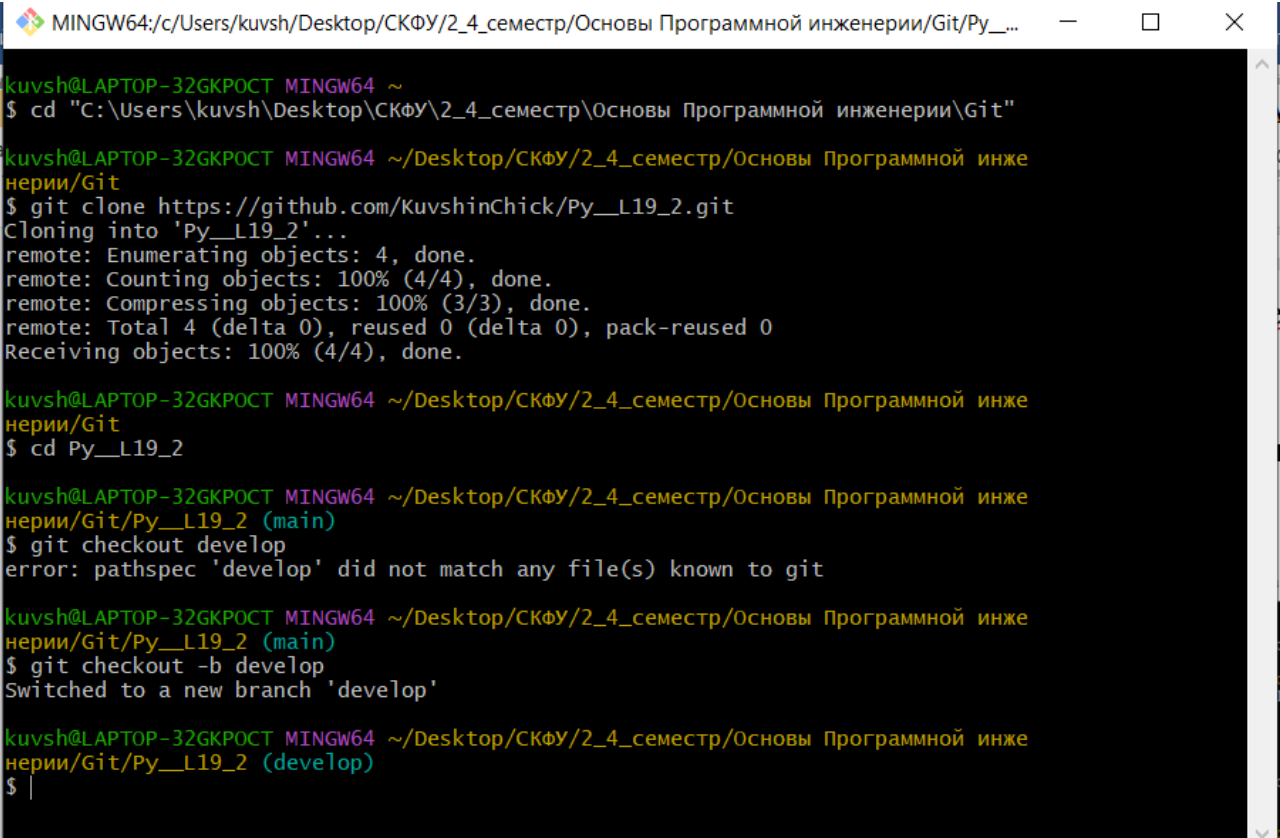
Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2023 г.

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

Ход работы:

1. Изучить теоретический материал работы.
2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python.
3. Выполните клонирование созданного репозитория.
4. Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm.
5. Организуйте свой репозиторий в соответствии с моделью ветвления git-flow.



```
MINGW64:/c:/Users/kuvsh/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git/Py_...
kuvsh@LAPTOP-32GKPOCT MINGW64 ~
$ cd "C:\Users\kuvsh\Desktop\СКФУ\2_4_семестр\Основы Программной инженерии\Git"

kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git
$ git clone https://github.com/KuvshinChick/Py__L19_2.git
Cloning into 'Py__L19_2'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (4/4), done.

kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git
$ cd Py__L19_2

kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git/Py__L19_2 (main)
$ git checkout develop
error: pathspec 'develop' did not match any file(s) known to git

kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git/Py__L19_2 (main)
$ git checkout -b develop
Switched to a new branch 'develop'

kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git/Py__L19_2 (develop)
$ |
```

Рисунок 19_2.1 – Клонирование репозитория и создание ветки develop

```
MINGW64/c:/Users/kuvsh/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git/Py__...
kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git/Py__L19_2 (develop)
$ git status
On branch develop
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore

no changes added to commit (use "git add" and/or "git commit -a")

kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git/Py__L19_2 (develop)
$ git add .

kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git/Py__L19_2 (develop)
$ git commit -m "modified .gitignore & readme"
[develop d34f2aa] modified .gitignore & readme
2 files changed, 133 insertions(+), 1 deletion(-)
create mode 100644 .gitignore

kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git/Py__L19_2 (develop)
$ |
```

Рисунок 19_2.2 – Обновление .gitignore и readme

6. Создайте проект PyCharm в папке репозитория.

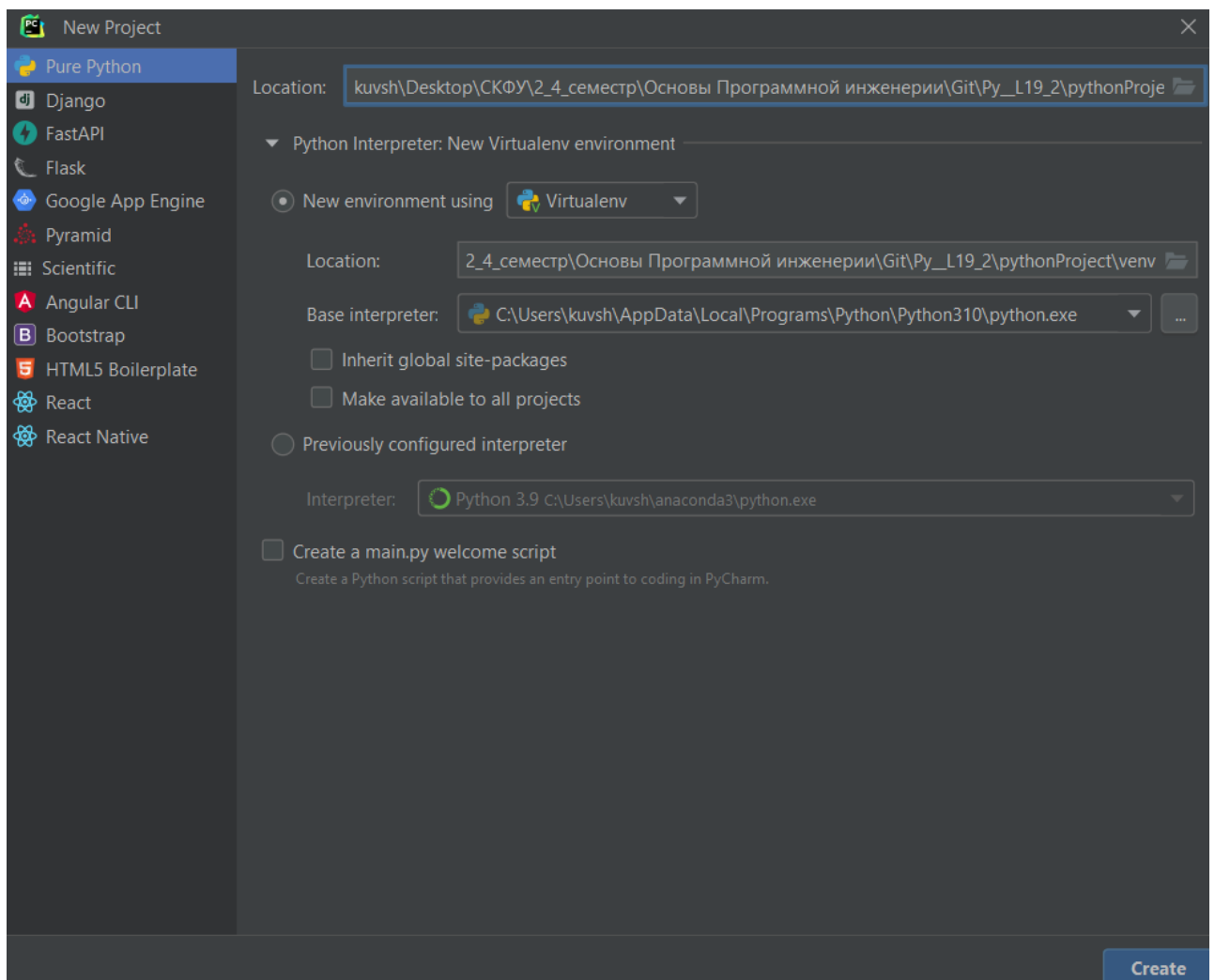
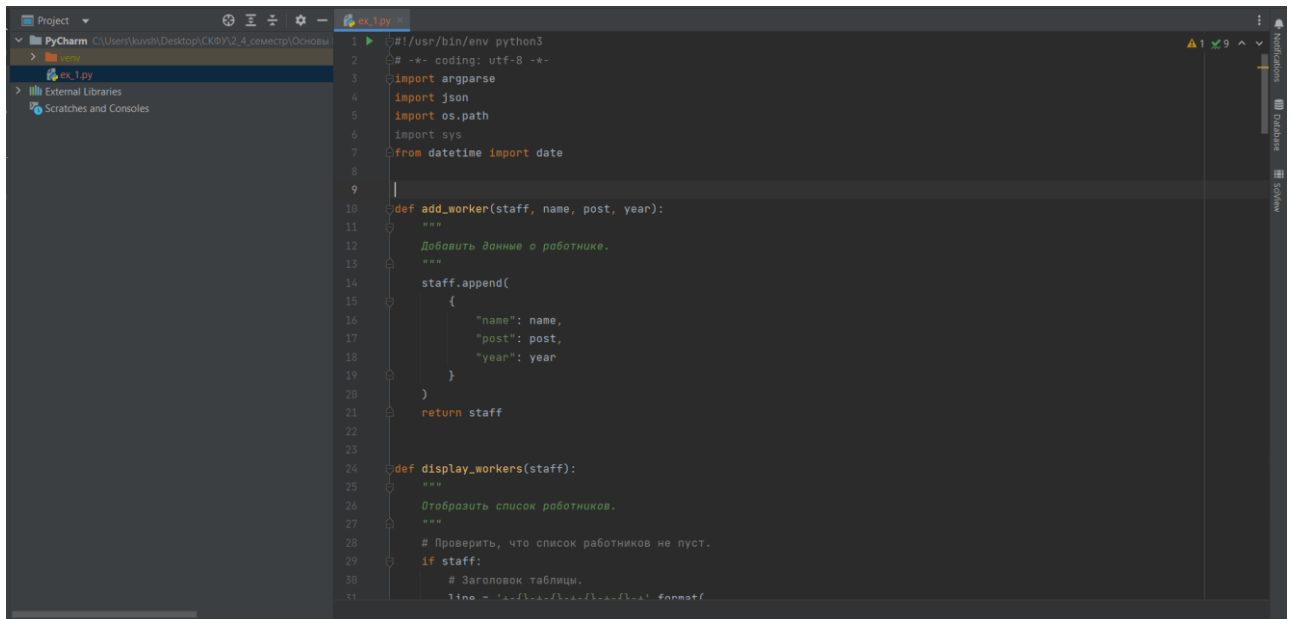


Рисунок 19_2.3 – Создание проекта и виртуального окружения

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

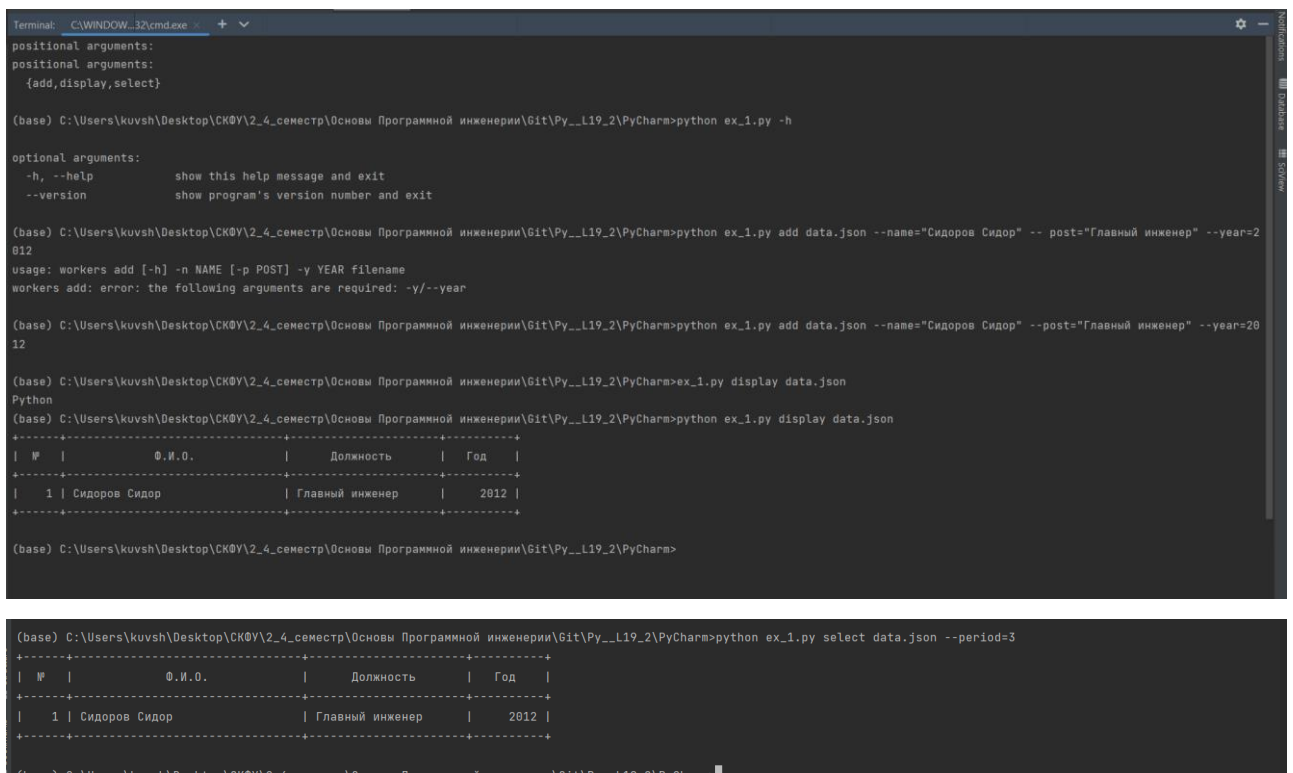
8. Приведите в отчете скриншоты результатов выполнения примера при различных исходных данных вводимых с клавиатуры.

Пример 1. Для примера 1 лабораторной работы 2.16 разработайте интерфейс командной строки.



```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 import argparse
4 import json
5 import os.path
6 import sys
7 from datetime import date
8
9
10 def add_worker(staff, name, post, year):
11     """
12     Добавить данные о работнике.
13     """
14     staff.append(
15         {
16             "name": name,
17             "post": post,
18             "year": year
19         }
20     )
21     return staff
22
23
24 def display_workers(staff):
25     """
26     Отобразить список работников.
27     """
28     # Проверить, что список работников не пуст.
29     if staff:
30         # Заголовок таблицы.
31         line = '%-10s%-10s%-10s%-10s' % format(
```

Рисунок 19_2.4 – Проработка примера



```
Terminal: C:\WINDOWS\system32\cmd.exe
positional arguments:
positional arguments:
{add,display,select}

(base) C:\Users\kuvsh\Desktop\СКФ\2_4_семестр\Основы Программной инженерии\Git\Py_L19_2\PyCharm>python ex_1.py -h

optional arguments:
-h, --help            show this help message and exit
--version             show program's version number and exit

(base) C:\Users\kuvsh\Desktop\СКФ\2_4_семестр\Основы Программной инженерии\Git\Py_L19_2\PyCharm>python ex_1.py add data.json --name="Сидоров Сидор" -- post="Главный инженер" --year=2012
usage: workers add [-h] -n NAME [-p POST] -y YEAR filename
workers add: error: the following arguments are required: -y/--year

(base) C:\Users\kuvsh\Desktop\СКФ\2_4_семестр\Основы Программной инженерии\Git\Py_L19_2\PyCharm>python ex_1.py add data.json --name="Сидоров Сидор" --post="Главный инженер" --year=2012

(base) C:\Users\kuvsh\Desktop\СКФ\2_4_семестр\Основы Программной инженерии\Git\Py_L19_2\PyCharm>python ex_1.py display data.json
Python
(base) C:\Users\kuvsh\Desktop\СКФ\2_4_семестр\Основы Программной инженерии\Git\Py_L19_2\PyCharm>python ex_1.py display data.json
+-----+-----+-----+-----+
| № |      Ф.И.О.      | Должность | Год |
+-----+-----+-----+-----+
| 1 | Сидоров Сидор    | Главный инженер | 2012 |
+-----+-----+-----+-----+

(base) C:\Users\kuvsh\Desktop\СКФ\2_4_семестр\Основы Программной инженерии\Git\Py_L19_2\PyCharm>

(base) C:\Users\kuvsh\Desktop\СКФ\2_4_семестр\Основы Программной инженерии\Git\Py_L19_2\PyCharm>python ex_1.py select data.json --period=3
+-----+-----+-----+-----+
| № |      Ф.И.О.      | Должность | Год |
+-----+-----+-----+-----+
| 1 | Сидоров Сидор    | Главный инженер | 2012 |
+-----+-----+-----+-----+

(base) C:\Users\kuvsh\Desktop\СКФ\2_4_семестр\Основы Программной инженерии\Git\Py_L19_2\PyCharm>
```

Рисунок 19_2.5 – Результат проработки программы

Индивидуальное задание.

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

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`.

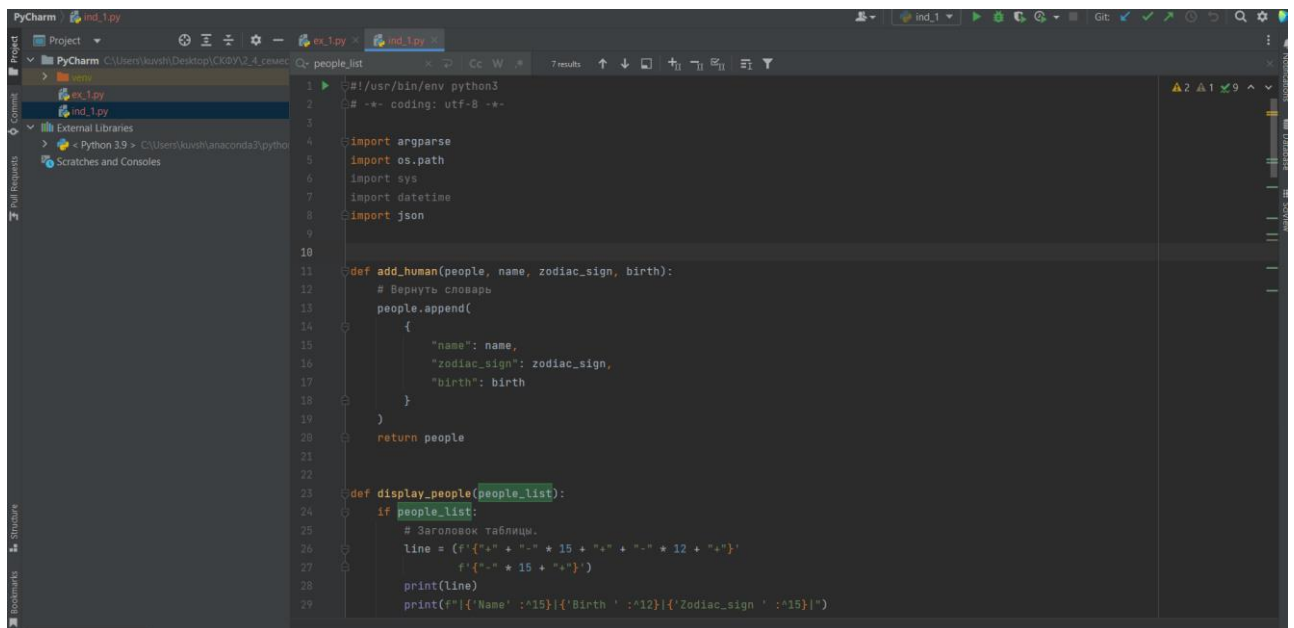
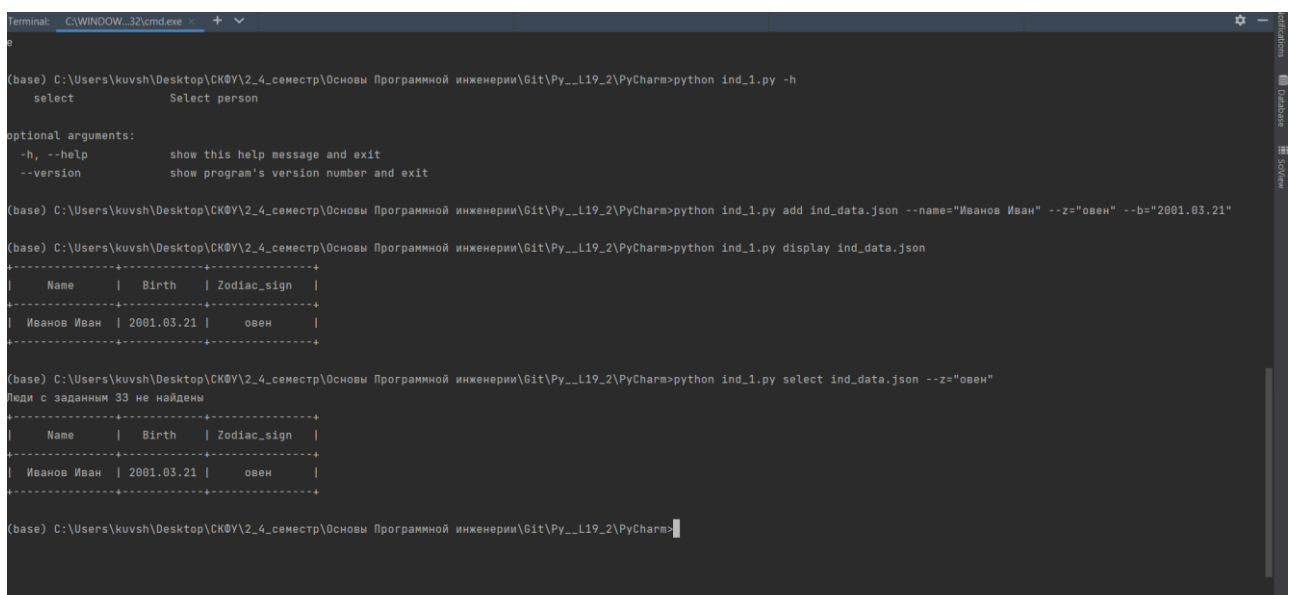


Рисунок 19_2.7 – Проработка индивидуального задания



```
(base) C:\Users\kuvsh\Desktop\СКФV\2_4_семестр\Основы Программной инженерии\Git\Py__L19_2\PyCharm>python ind_1.py select ind2.txt --z="овен"
Люди с заданным ЗЗ не найдены

+-----+
| Name | Birth | Zodiac_sign |
+-----+
| Иванченко И.И. | 2001.03.21 | овен |
+-----+
```

(base) C:\Users\kuvsh\Desktop\СКФV\2_4_семестр\Основы Программной инженерии\Git\Py__L19_2\PyCharm>

Рисунок 19_2.8 – Результат проработки программы

```
terminal: C:\WINDOWS\system32\cmd.exe
(base) C:\Users\kuvsh\Desktop\СКФV\2_4_семестр\Основы Программной инженерии\Git\Py__L19_2>cd PyCharm
(base) C:\Users\kuvsh\Desktop\СКФV\2_4_семестр\Основы Программной инженерии\Git\Py__L19_2\PyCharm>python ind_2_click.py display ind_data_2_click.json
Список пуст.

(base) C:\Users\kuvsh\Desktop\СКФV\2_4_семестр\Основы Программной инженерии\Git\Py__L19_2\PyCharm>python ind_2_click.py add ind_data_2_click.json
Please enter person's name: Ари
Please enter person's zodiac_sign: лев
Please enter person's birth: 2006.08.23

(base) C:\Users\kuvsh\Desktop\СКФV\2_4_семестр\Основы Программной инженерии\Git\Py__L19_2\PyCharm>python ind_2_click.py display ind_data_2_click.json

+-----+
| Name | Birth | Zodiac_sign |
+-----+
| Ари | 2006.08.23 | лев |
+-----+
```

(base) C:\Users\kuvsh\Desktop\СКФV\2_4_семестр\Основы Программной инженерии\Git\Py__L19_2\PyCharm>python ind_2_click.py select ind_data_2_click.json

Please enter person's zodiac_sign: лев

```
+-----+
| Name | Birth | Zodiac_sign |
+-----+
| Ари | 2006.08.23 | лев |
+-----+
```

(base) C:\Users\kuvsh\Desktop\СКФV\2_4_семестр\Основы Программной инженерии\Git\Py__L19_2\PyCharm>

Рисунок 19_2.9 – Результат проработки программы

9. Зафиксируйте сделанные изменения в репозитории.
10. Приведите в отчете скриншоты работы программ решения индивидуальных заданий.
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 (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 `args` – 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.

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)
```

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. Хотелось бы остановить на ней Ваше внимание.

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.