

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ  
АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО  
ОБРАЗОВАНИЯ «СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»  
ИНСТИТУТ ЦИФРОВОГО РАЗВИТИЯ**

**Отчет по лабораторной работе № 2.17**

**«Работа с данными формата JSON в языке Python»**

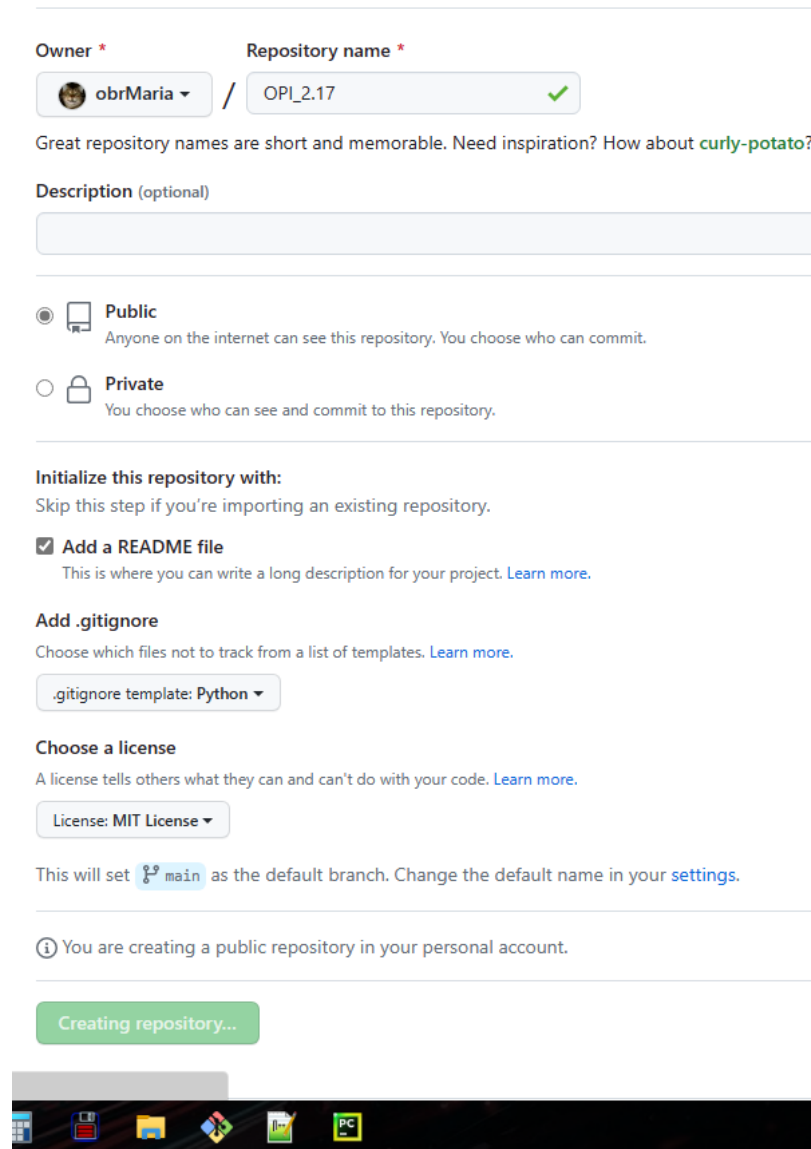
**по дисциплине «Основы программной инженерии»**

Выполнила:  
Образцова Мария Дмитриевна,  
2 курс, группа ПИЖ-б-о-21-1,  
Проверил:  
Доцент кафедры инфокоммуникаций,  
Воронкин Р.А.

Ставрополь, 2023 г.

## 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.



Owner \* Repository name \*

obrMaria / OPI\_2.17

Great repository names are short and memorable. Need inspiration? How about [curly-potato?](#)

Description (optional)

☒ Public  
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](#).

*i* You are creating a public repository in your personal account.

Creating repository...

Рисунок 1 – создание репозитория

Рисунок 2 – клонирование репозитория

Рисунок 3 – сохранение изменений в файле gitignore

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

```
workers (1)

py ind2.py workers.py x

/bin/env python3
coding: utf-8 -*-

argparse __exception__: (<class 'AttributeE
json
os.path
sys
atetime import date

d_worker(staff, name, post, year):
"
бавить данные о работнике.
"
aff.append(
{
    "name": name,
    "post": post,
    "year": year
}

turn staff

;

splay_workers(staff):
"
образить список работников.
"
Проверить, что список работников не пуст.
staff:
    # Заголовок таблицы.
    line = '+-{}--{}--{}--{}--'.format(
        '-' * 4,
```

```
data.json x
1  [
2      {
3          "name": "Сидоров Сидор",
4          "post": "Главный инженер",
5          "year": 2012
6      }
7  ]
```

### Индивидуальные задания

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

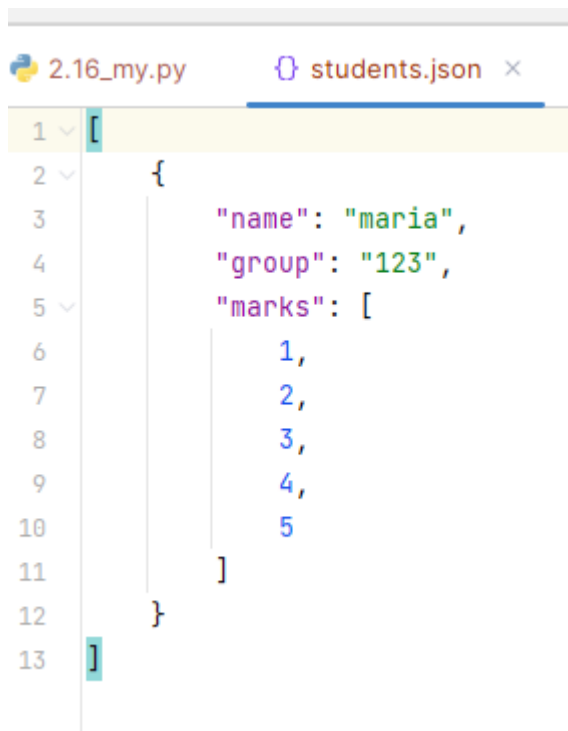
```
C:\Users\M\Desktop\PY\2,17>python 2.16_my.py add students.json -n="maria" -g="123" -m="1 2 3 4 5"
[{'name': 'maria', 'group': '123', 'marks': [1, 2, 3, 4, 5]}]

C:\Users\M\Desktop\PY\2,17>python 2.16_my.py display students.json
+-----+-----+-----+-----+
| № | Ф.И.О. | Группа | Оценки |
+-----+-----+-----+-----+
| 1 | maria | 123 | 1,2,3,4,5 |
+-----+-----+-----+-----+

C:\Users\M\Desktop\PY\2,17>python 2.16_my.py find students.json
список студентов пуст
```

### 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



```
2.16_my.py  students.json x
1  [
2      {
3          "name": "maria",
4          "group": "123",
5          "marks": [
6              1,
7              2,
8              3,
9              4,
10             5
11         ]
12     }
13 ]
```

## Вопросы для защиты работы

### 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 – компьютер с клавиатурой и монитором.

### 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` 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` ?

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)
```

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 `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.

Для начала работы с `argparse` необходимо задать парсер. 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`.

В нашем случае:

```
>>> print(parser.parse_args(['-n', '3'])) Namespace(n='3')
>>> print(parser.parse_args([])) Namespace(n=None)
```

```
>>> print(parser.parse_args(['-a', '3'])) error: unrecognized arguments: -a 3
```

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.

В зависимости от переданного в конструктор парсера аргумента add\_help (булевого типа), будет определяться, включать или не включать в стандартный вывод по ключам ['-h', '--help'] сообщения о помощи. То же самое будет иметь место с аргументом version (строкового типа), ключи по умолчанию: ['-v', '--version']. При запросе помощи или номера версии, дальнейшее выполнение прерывается.

```
parser = argparse.ArgumentParser(add_help=True, version='4.0')
```