

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 №20
дисциплины «Основы программной инженерии»

Выполнил:
Матвеев Александр Иванович
2 курс, группа ПИЖ-б-о-22-1,
09.03.04 «Программная инженерия»,
направленность (профиль) «Разработка и
сопровождение программного
обеспечения», очная форма обучения

(подпись)

Проверил Воронкин Роман Александрович

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2024 г.

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

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

Ход работы.

1. Создание нового репозитория с лицензией MIT.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk ().*

Owner * SashkaHacker / **Repository name *** laba10
✔ laba10 is available.

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

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:

☐ **Add a README file**
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore
.gitignore template: Python

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license
License: MIT License

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

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

Create repository

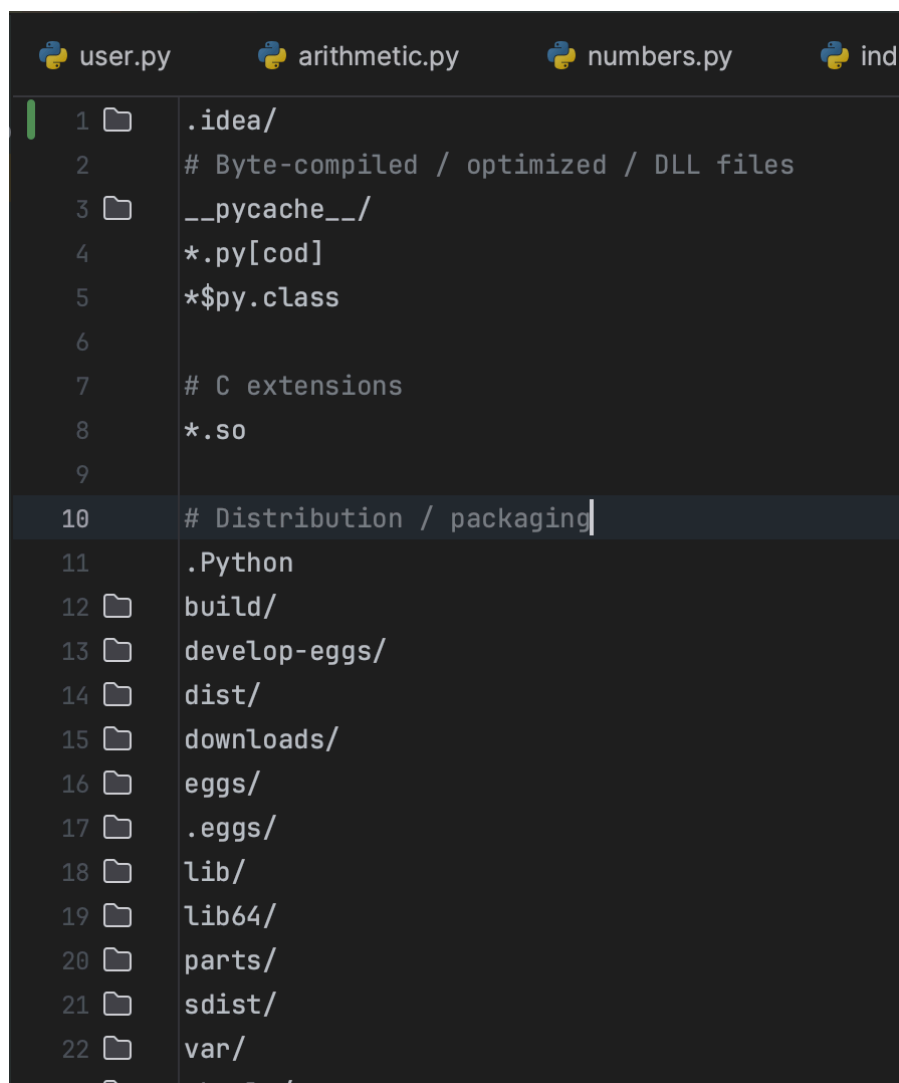
Рисунок 1 – Создание репозитория

2. Клонировал репозиторий на рабочий ПК.

```
[→ GitHub git clone https://github.com/SashkaHacker/laba16.git
Cloning into 'laba16'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), done.
```

Рисунок 2 – Клонирование репозитория

3. Дополнил файл .gitignore необходимыми инструкциями.

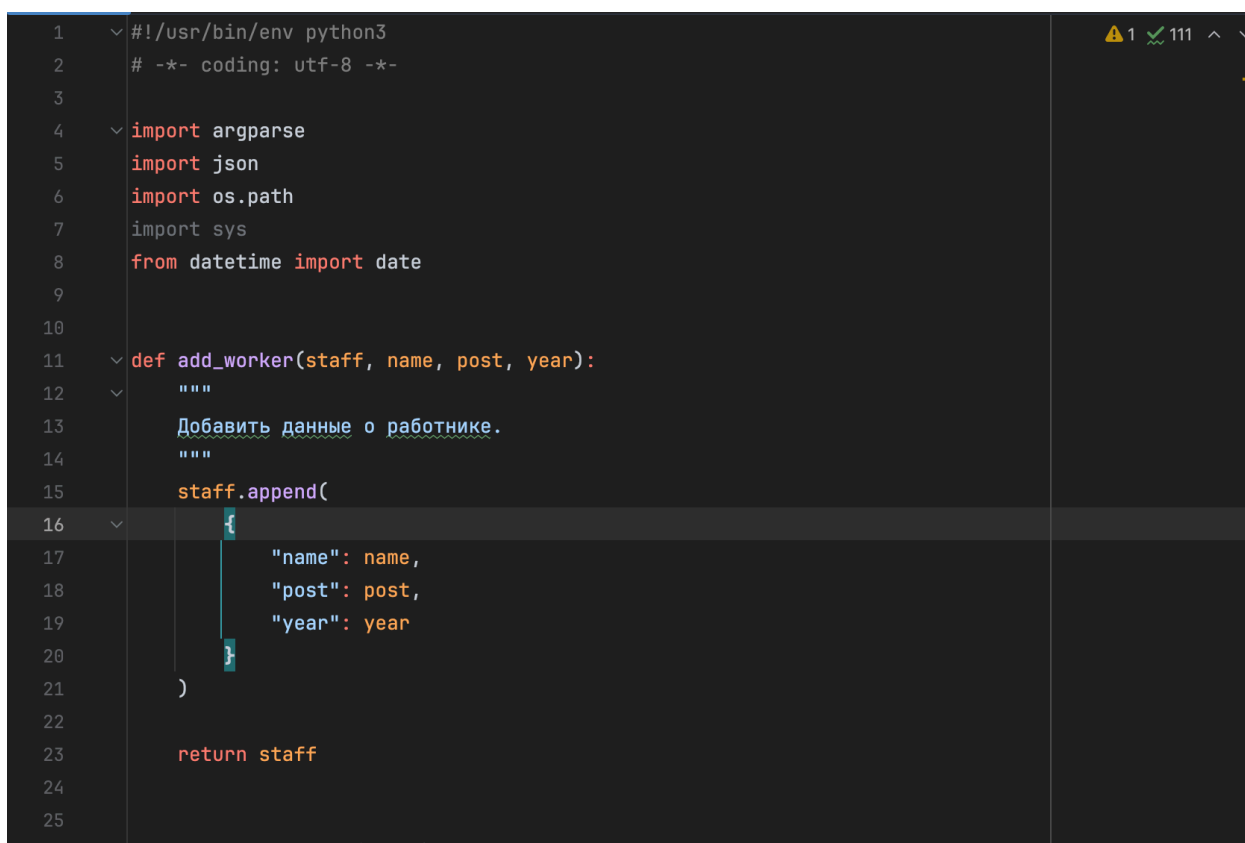


The screenshot shows a code editor with a dark theme. At the top, there are four tabs: 'user.py', 'arithmetic.py', 'numbers.py', and 'ind'. The active tab is 'user.py'. The editor displays the content of the '.gitignore' file, which is numbered from 1 to 22. The content includes comments and file/folder names to be ignored:

```
1  .idea/
2  # Byte-compiled / optimized / DLL files
3  __pycache__/
4  *.py[cod]
5  *$py.class
6
7  # C extensions
8  *.so
9
10 # Distribution / packaging
11 .Python
12 build/
13 develop-eggs/
14 dist/
15 downloads/
16 eggs/
17 .eggs/
18 lib/
19 lib64/
20 parts/
21 sdist/
22 var/
```

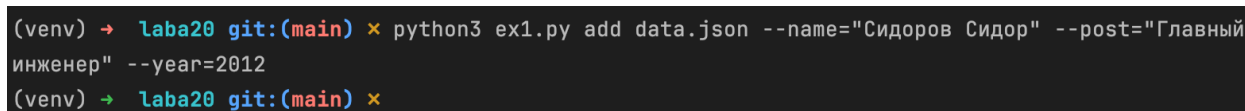
Рисунок 3 – Файл .gitignore

4. Проработка примера из лабораторной работы.

A screenshot of a code editor showing a Python script. The script starts with a shebang line and a UTF-8 encoding declaration. It imports argparse, json, os.path, sys, and date from datetime. A function add_worker is defined, which takes staff, name, post, and year as arguments. It appends a dictionary with the worker's details to the staff list and returns it. The code is as follows:

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  import argparse
5  import json
6  import os.path
7  import sys
8  from datetime import date
9
10
11 def add_worker(staff, name, post, year):
12     """
13     Добавить данные о работнике.
14     """
15     staff.append(
16         {
17             "name": name,
18             "post": post,
19             "year": year
20         }
21     )
22
23     return staff
```

Рисунок 4 – Пример №1

A screenshot of a terminal window showing two commands being executed in a virtual environment. The first command runs a Python script with specific arguments, and the second command shows the current directory and git status. The commands are:

```
(venv) → laba20 git:(main) × python3 ex1.py add data.json --name="Сидоров Сидор" --post="Главный инженер" --year=2012
(venv) → laba20 git:(main) ×
```

Рисунок 5 – Входные данные программы

```
1  ✓ [
2  ✓   {
3      "name": "Иванов Иван",
4      "post": "Директор",
5      "year": 2007
6  },
7  ✓   {
8      "name": "Петров Петр",
9      "post": "Бухгалтер",
10     "year": 2010
11  },
12  ✓   {
13     "name": "Сидоров Сидор",
14     "post": "Главный инженер",
15     "year": 2012
16  },
17  ✓   {
18     "name": "Mark Markdown",
19     "post": "Main engeneer",
20     "year": 2012
21  },
22  ✓   {
23     "name": "Сидоров Сидор",
24     "post": "Главный инженер",
25     "year": 2012
26  }
27  ]
```

Рисунок 6 – Результат исполнения программы

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

```
102
103  def main(command_line=None):
104      # Создать родительский парсер для определения имени файла.
105      file_parser = argparse.ArgumentParser(add_help=False)
106      file_parser.add_argument(
107          *name_or_flags: "filename", action="store", help="The data file name"
108      )
109
110      # Создать основной парсер командной строки.
111      parser = argparse.ArgumentParser(description="workers")
112      parser.add_argument(*name_or_flags: "--version", action="version", version="%prog s 0.1.0")
113      subparsers = parser.add_subparsers(dest="command")
114
115      # Создать субпарсер для добавления работника.
116      add = subparsers.add_parser(
117          name: "add", parents=[file_parser], help="Add a new worker"
118      )
119      add.add_argument(
120          *name_or_flags: "-n", "--name", action="store", required=True, help="Имя работника"
121      )
122      add.add_argument(
123          *name_or_flags: "-s", "--surname", action="store", help="Фамилия работника"
124      )
125      add.add_argument(
126          *name_or_flags: "-p",
127          "--phone",
128          action="store",
129          required=True,
130          help="Номер телефона работника",
131      )
```

Рисунок 7 – Код программы

```
(venv) → laba20 git:(main) × python3 individual.py select -p=3456789012 data.json
Фамилия: Сидоров
Имя: Сидор
Номер телефона: 3456789012
Дата рождения: 01:12:1978
```

Рисунок 8 – Пример работы программы

```
(venv) → laba20 git:(main) × python3 individual.py display data.json
```

№	Фамилия	Имя	Номер телефона	Дата рождения
1	Сидоров	Сидор	3456789012	01:12:1978
2	Петров	Петр	2345678901	15:06:1985
3	Николаев	Николай	5678901234	11:11:1988
4	Иванов	Иван	1234567890	28:02:1990
5	Алексеев	Алексей	4567890123	20:05:1992

Рисунок 9 – Пример работы программы

6. Выполнение индивидуального задания 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 click
import json
from datetime import datetime
from validation import ListWorkers

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

@cli.command()
@click.argument('filename')
@click.argument('surname')
@click.argument('name')
@click.argument(*param_decls: 'phone', type=int)
@click.argument('date')
def add(filename, surname, name, phone, date):
    lst = load_workers(filename)
    add_worker(lst, surname, name, phone, date)
    save_workers(filename, lst)

@cli.command()
@click.argument('filename')
def display(filename):
    lst = load_workers(filename)
    show_workers(lst)
```

Рисунок 10 – Код программы

```
(venv) → lab20 git:(main) × python3 individual_hard.py display data.json
```

№	Фамилия	Имя	Номер телефона	Дата рождения
1	Сидоров	Сидор	3456789012	01:12:1978
2	Петров	Петр	2345678901	15:06:1985
3	Николаев	Николай	5678901234	11:11:1988
4	Иванов	Иван	1234567890	28:02:1990
5	Алексеев	Алексей	4567890123	20:05:1992

```
(venv) → lab20 git:(main) ×
```

Рисунок 11 – Пример работы программы


```
(venv) → laba20 git:(main) ✕ python3 individual_hard.py select data.json 4567890123
Фамилия: Алексеев
Имя: Алексей
Номер телефона: 4567890123
Дата рождения: 20:05:1992
(venv) → laba20 git:(main) ✕
```

Рисунок 12 – Пример работы программы

Контрольные вопросы:

1. Отличие терминала и консоли:

Консоль — это физическое устройство или его программный эмулятор, используемый для ввода и вывода данных. В историческом контексте консолью часто называли устройство с клавиатурой и монитором, подключенное к компьютеру.

Терминал — это программа (терминальный эмулятор), которая предоставляет пользователю текстовый интерфейс для взаимодействия с операционной системой через командную строку. В современном контексте термины "консоль" и "терминал" часто используются как синонимы.

2. Консольное приложение:

Консольное приложение — это программа, которая работает в текстовом режиме и взаимодействует с пользователем через командную строку (терминал). Пользователь управляет таким приложением с помощью текстовых команд, а программа выводит результаты своей работы также в текстовом виде.

3. Средства Python для построения приложений командной строки:

Модуль `sys` предоставляет доступ к некоторым переменным и функциям, работающим с интерпретатором Python. Например, `sys.argv` используется для получения аргументов командной строки.

Модуль `argparse` позволяет легко создавать пользовательские интерфейсы командной строки, обеспечивая обработку аргументов командной строки.

Модуль `getopt` — это более старый способ парсинга аргументов командной строки, похожий на стиль C-библиотеки `getopt`.

Библиотеки сторонних разработчиков, такие как `click` и `docopt`, также популярны для создания CLI.

4. Особенности построения CLI с использованием модуля `sys`:

Модуль `sys` не предоставляет встроенных средств для парсинга аргументов; он только дает доступ к списку аргументов командной строки через `sys.argv`.

Разработчику самому нужно обрабатывать `sys.argv` и реализовывать логику разбора и валидации аргументов.

5. Особенности построения CLI с использованием модуля `getopt`:

`Getopt` поддерживает базовые сценарии разбора аргументов, включая короткие и длинные опции (например, `-h` и `--help`). Модуль `getopt` может быть менее удобен и гибок по сравнению с более современными альтернативами, такими как `argparse`.

6. Особенности построения CLI с использованием модуля `argparse`:

`Argparse` — это мощный модуль для создания интерфейсов командной строки, который поддерживает создание сложных пользовательских интерфейсов с множеством опций и аргументов. Он позволяет легко добавлять аргументы и опции, устанавливать значения по умолчанию, проверять типы данных и генерировать сообщения.