

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 №2.6
дисциплины «Основы программной инженерии»

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

(подпись)

Руководитель практики:
Богданов С.С., ассистент кафедры
инфокоммуникаций

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2023 г.

Тема: Работа со словарями в языке Python.

Цель работы: приобретение навыков по работе со словарями при написании программ с помощью языка программирования Python версии 3.x.

Ход выполнения работы:

1. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и .gitignore файл для языка программирования Python:

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 ().*

Repository template

No template ▾

Start your repository with a template repository's contents.

Owner * PoTtaTto ▾ / **Repository name *** fundamentals_of_software

✔ fundamentals_of_software_engineering_lab2_5 is available.

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

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 ▾

Рисунок 1 – Создание репозитория с заданными настройками

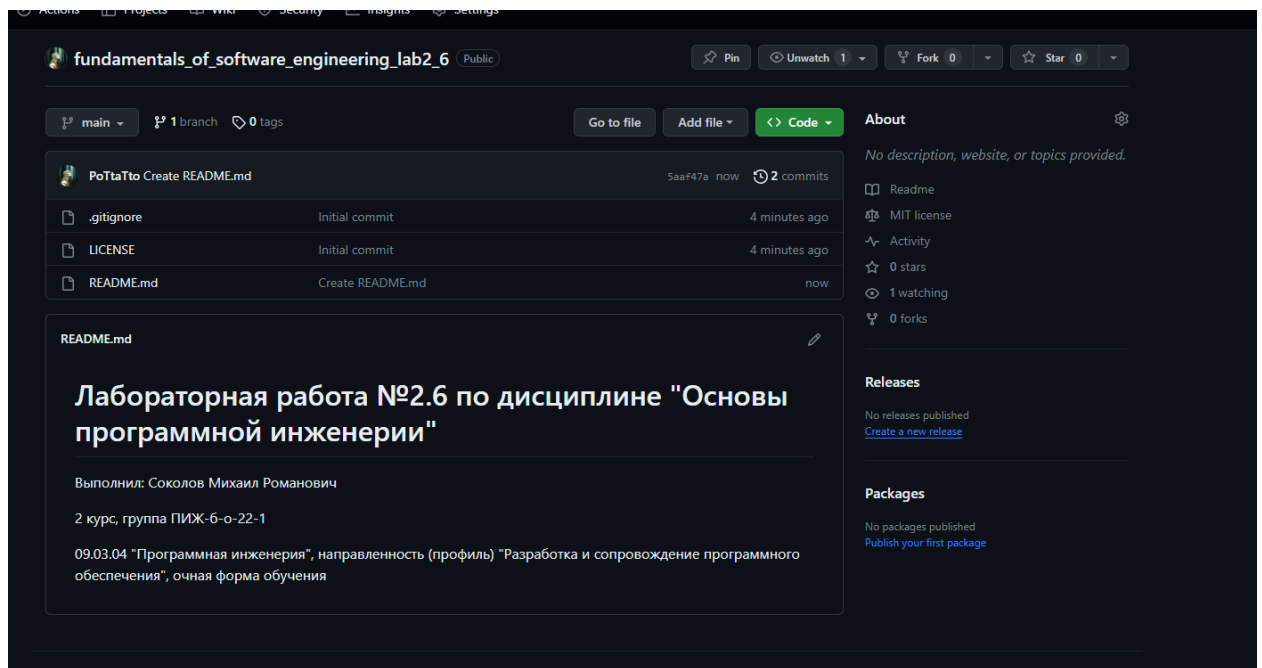


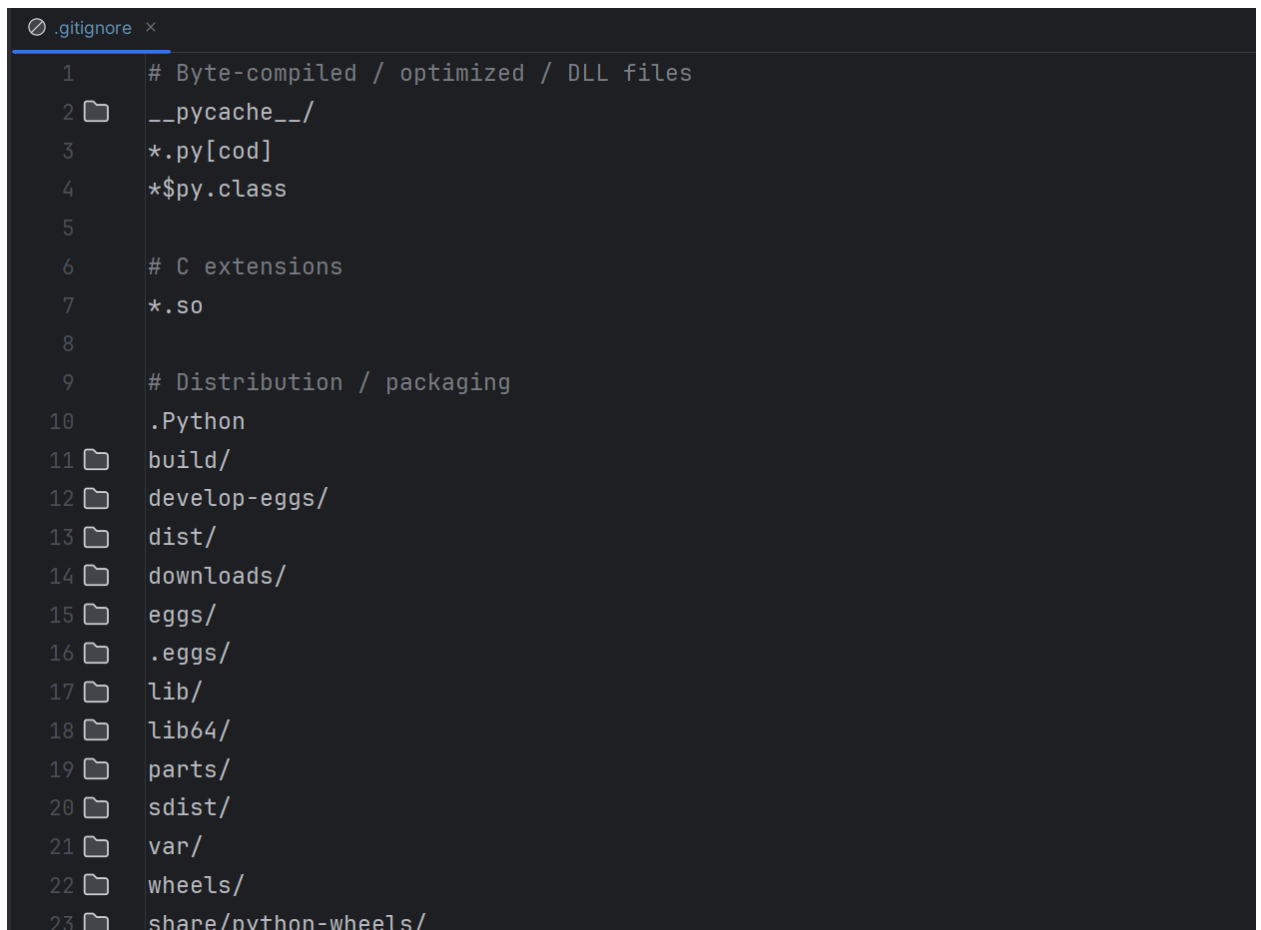
Рисунок 2 – Созданный репозиторий

```
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.6> git clone https://github.com/PoTtaTto/fundamentals_of_software_engineering_lab2_6
Cloning into 'fundamentals_of_software_engineering_lab2_6'...
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 7 (delta 1), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (7/7), done.
Resolving deltas: 100% (1/1), done.
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.6> |
```

Рисунок 3 – Клонирование репозитория

```
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.6> git checkout -b develop
Switched to a new branch 'develop'
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.6>
```

Рисунок 4 – Создание ветки develop, где будут происходить изменения проекта до его полного релиза



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

Рисунок 5 – Часть .gitignore файла, созданного GitHub

2. Проработаем примеры лабораторной работы, фиксируя изменения.

Создадим для каждого примера отдельный модуль языка Python:

```
1. #!/usr/bin/env python3
2. # -*- coding: utf-8 -*-
3.
4. import sys
5. from datetime import date
6.
7.
8. if __name__ == '__main__':
9.     workers = []
10.     while True:
11.         command = input(">>> ").lower()
12.
13.         cmd_parts = command.split(maxsplit=1)
14.         match cmd_parts[0]:
15.             case 'exit':
16.                 break
17.             case 'add':
```

```

18.         name = input('Введите фамилию и инициалы: ')
19.         post = input('Введите должность: ')
20.         year = int(input('Введите год поступления: '))
21.         workers.append({
22.             'name': name,
23.             'post': post,
24.             'year': year
25.         })
26.         if len(workers) > 1:
27.             workers.sort(key=lambda item: item.get('name', ''))
28.     case 'list':
29.         line = f'+-{"-" * 4}+-{"-" * 30}+-{"-" * 20}+-{"-" * 8}+-'
30.         print(line)
31.         header = f"| {'№':^4} | {'Ф.И.О.':^30} | {'Должность':^20} |
32.         {'Год':^8} |"
33.         print(header)
34.         print(line)
35.         for index, worker in enumerate(workers, 1):
36.             name, post, year = worker.get('name', ''),
37.             worker.get('post', ''), worker.get('year', 0)
38.             recording = f"| {index:^4} | {name:^30} | {post:^20} |
39.             {year:^8} |"
40.             print(recording)
41.         print(line)
42.     case 'select':
43.         today = date.today()
44.         period = int(cmd_parts[1])
45.         count = 0
46.         for worker in workers:
47.             if today.year - worker.get('year', today.year) >= period:
48.                 count += 1
49.                 print(f'{count:^4}: {worker.get("name", "")}')
50.         if count == 0:
51.             print('Работники с заданным стажем не найдены.')
52.     case 'help':
53.         print("Список команд:\n")
54.         print("add - добавить работника;")
55.         print("list - вывести список работников;")
56.         print("select <стаж> - запросить работников со стажем;")
57.         print("help - отобразить справку;")
58.         print("exit - завершить работу с программой.")
59.     case _:
60.         print(f"Неизвестная команда {command}", file=sys.stderr)

```

Листинг 1 – код примера из лабораторной работы

```
инженерии\Лабораторная работа 2.6\fundamentals_of_software_eng
>>> help
Список команд:

add - добавить работника;
list - вывести список работников;
select <стаж> - запросить работников со стажем;
help - отобразить справку;
exit - завершить работу с программой.
>>> |
```

Рисунок 6 – Вывод программы в консоль. Использование команды help

```
>>> exit

Process finished with exit code 0
```

Рисунок 7 – Вывод программы в консоль. Использование команды exit

```
>>> asdf
>>> Неизвестная команда asdf
```

Рисунок 8 – Вывод программы в консоль. Использование неизвестной команды

```
>>> add
Введите фамилию и инициалы: Соколов М.Р.
Введите должность: Студент
Введите год поступления: 2022
>>> add
Введите фамилию и инициалы: Пупки В.И
Введите должность: Студент
Введите год поступления: 2020
>>>
```

Рисунок 9 – Вывод программы в консоль. Использование команды add

```
>>> list
```

№	Ф.И.О.	Должность	Год
1	Пупки В.И	Студент	2020
2	Соколов М.Р.	Студент	2022

Рисунок 10 – Вывод программы в консоль. Использование команды list

```
>>> select 1
1 : Пупкин В.И.
2 : Соколов М.Р.
>>> |
```

Рисунок 11 – Вывод программы в консоль. Использование команды select со
стажем от 1-го года

```
>>> select 2
1 : Пупкин В.И.
>>> |
```

Рисунок 12 – Вывод программы в консоль. Использование команды select со стажем от 2-х лет

```
>>> select 5
Работники с заданным стажем не найдены.
>>>
```

Рисунок 13 – Вывод программы в консоль. Использование команды select со стажем от 5 лет

3. Решить задачу: создайте словарь, связав его с переменной `school` , и наполните данными, которые бы отражали количество учащихся в разных классах (1а, 1б, 2б, 6а, 7в и т. п.). Внесите изменения в словарь согласно следующему: а) в одном из классов изменилось количество учащихся, б) в школе появился новый класс, с) в школе был расформирован (удален) другой класс. Вычислите общее количество учащихся в школе. Зафиксировать изменения в репозитории:

```
1. #!/usr/bin/env python3
2. # -*- coding: utf-8 -*-
3.
4. if __name__ == '__main__':
5.     school = {
6.         '1а': 30, '1б': 25,
7.         '2б': 30, '6а': 12,
8.         '7в': 5
9.     }
10.    print(school)
11.
12.    # Изменение количество учащихся в классе 2б
13.    school['2б'] -= 3
14.    # Появление нового класса 11б в школе
```



```

15.     school['11б'] = 17
16.     # Расформирование 7в класса
17.     del school['7в']
18.     # Общее количество учащихся
19.     general_count = sum(school.values())
20.
21.     print(school)

```

Листинг 2 – Код задачи

```

{'1а': 30, '1б': 25, '2б': 30, '6а': 12, '7в': 5}
{'1а': 30, '1б': 25, '2б': 27, '6а': 12, '11б': 17}

```

Рисунок 14 – Вывод в консоль

4. Решить задачу: создайте словарь, где ключами являются числа, а значениями – строки. Примените к нему метод `items()`, с помощью полученного объекта `dict_items` создайте новый словарь, "обратный" исходному, т. е. ключами являются строки, а значениями – числа. Зафиксировать изменения в репозитории.

The screenshot shows a code editor with a file named `task2.py`. The code defines a dictionary `num_dict` with numeric keys and string values, converts it to `dict_items`, and then creates a new dictionary `str_dict` with string keys and numeric values. The console output shows the execution of the code, displaying the original dictionary, the `dict_items` object, and the resulting reversed dictionary.

```

task2.py x
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  if __name__ == '__main__':
5      num_dict = {1: 'one', 2: 'two', 3: 'three'}
6      print(num_dict)
7      dict_items = num_dict.items()
8      str_dict = {value: key for key, value in dict_items}
9      print(str_dict)

```

Run manual_task1 x task2 x

```

C:\Users\user\Documents\fundamentals_of_software_engineering_lab2_6\task2.py
{1: 'one', 2: 'two', 3: 'three'}
{'one': 1, 'two': 2, 'three': 3}
Process finished with exit code 0

```

Рисунок 15 – Код задачи и вывод в консоль

5. Решить индивидуальное задание, построить UML-диаграмму и зафиксировать изменения в репозитории. Задача: использовать словарь, содержащий следующие ключи: название пункта назначения; номер поезда; время отправления. Написать программу, выполняющую следующие действия: ввод с клавиатуры данных в список, состоящий из словарей заданной структуры; записи должны быть упорядочены по времени отправления поезда; вывод на экран информации о поездах, направляющихся в пункт, название которого введено с клавиатуры; если таких поездов нет, выдать на дисплей соответствующее сообщение:

```
1. #!/usr/bin/env python3
2. # -*- coding: utf-8 -*-
3.
4. import sys
5. from random import randint
6.
7.
8. if __name__ == '__main__':
9.     trains = []
10.    while True:
11.        cmd = input('>>> ')
12.        cmd_parts = cmd.split(maxsplit=1)
13.        match cmd_parts[0]:
14.            case 'add':
15.                train_num = int(input('Введите номер поезда: '))
16.                destination = input('Введите пункт назначения: ')
17.                start_time = input('Введите время выезда: ')
18.                trains.append({'num': train_num, 'destination': destination,
19.                               'start_time': start_time})
20.                if len(trains) > 1:
21.                    trains.sort(key=lambda item: item['start_time'])
22.            case 'list':
23.                line = f'+-{"-" * 15}-+{"-" * 30}-+{"-" * 25}-+'
24.                print(line)
25.                header = f'| {'№ поезда':^15} | {'Пункт назначения':^30} |
26.                {'Время отъезда':^25} |"
27.                print(header)
28.                print(line)
29.                for train in trains:
30.                    num = train.get('num', randint(1000, 10000))
31.                    destination = train.get('destination', 'None')
32.                    start_time = train.get('start_time', 'None')
```

```

31.             recording = f"| {num:^15} | {destination:^30} |
{start_time:^25} |"
32.             print(recording)
33.             print(line)
34.         case 'select':
35.             cmd_destination = cmd_parts[1]
36.             select_trains = [train for train in trains if
train['destination'] == cmd_destination]
37.             if len(select_trains) > 1:
38.                 for train in select_trains:
39.                     print(f'{train["num"]:^15}:
{train["start_time"]:^25}')
40.             else:
41.                 print('Нет поездов едущих в данное место!',
file=sys.stderr)
42.         case 'help':
43.             print("Список команд:\n")
44.             print("add - добавить поезд;")
45.             print("list - вывести список поездов;")
46.             print("select <пункт назначения> - запросить поезда с пунктом
назначения;")
47.             print("help - отобразить справку;")
48.             print("exit - завершить работу с программой.")
49.         case 'exit':
50.             break
51.         case _:
52.             print(f"Неизвестная команда {cmd}", file=sys.stderr)

```

Листинг 3 – Код индивидуального задания

```

>>> help
Список команд:

add - добавить поезд;
list - вывести список поездов;
select <пункт назначения> - запросить поезда с пунктом назначения;
help - отобразить справку;
exit - завершить работу с программой.

```

Рисунок 16 – Команда help

```
>>> exit
```

```
Process finished with exit code 0
```

Рисунок 17 – Команда exit

```
>>> add
```

Введите номер поезда: 1

Введите пункт назначения: Ставрополь

Введите время выезда: 2023-11-20 15:00

```
>>> add
```

Введите номер поезда: 2

Введите пункт назначения: Ставрополь

Введите время выезда: 2023-11-21 03:30

```
>>> add
```

Введите номер поезда: 3

Введите пункт назначения: Москва

Введите время выезда: 2024-01-01 00:00

```
>>> |
```

Рисунок 18 – Использование команды add

```
>>> list
```

№ поезда	Пункт назначения	Время отъезда
1	Ставрополь	2023-11-20 15:00
2	Ставрополь	2023-11-21 03:30
3	Москва	2024-01-01 00:00

```
>>>
```

Рисунок 19 – Использование команды list

```
select Ставрополь
      1      :      2023-11-20 15:00
      2      :      2023-11-21 03:30
>>>
```

Рисунок 20 – Использование команды select (1)

```
select Пятигорск
>>> Нет поездов едущих в данное место!
```

Рисунок 21 – Использование команды select (2)

```
dsfsadfsafd
Неизвестная команда dsfsadfsafd
>>>
```

Рисунок 22 – Использование неизвестной команды

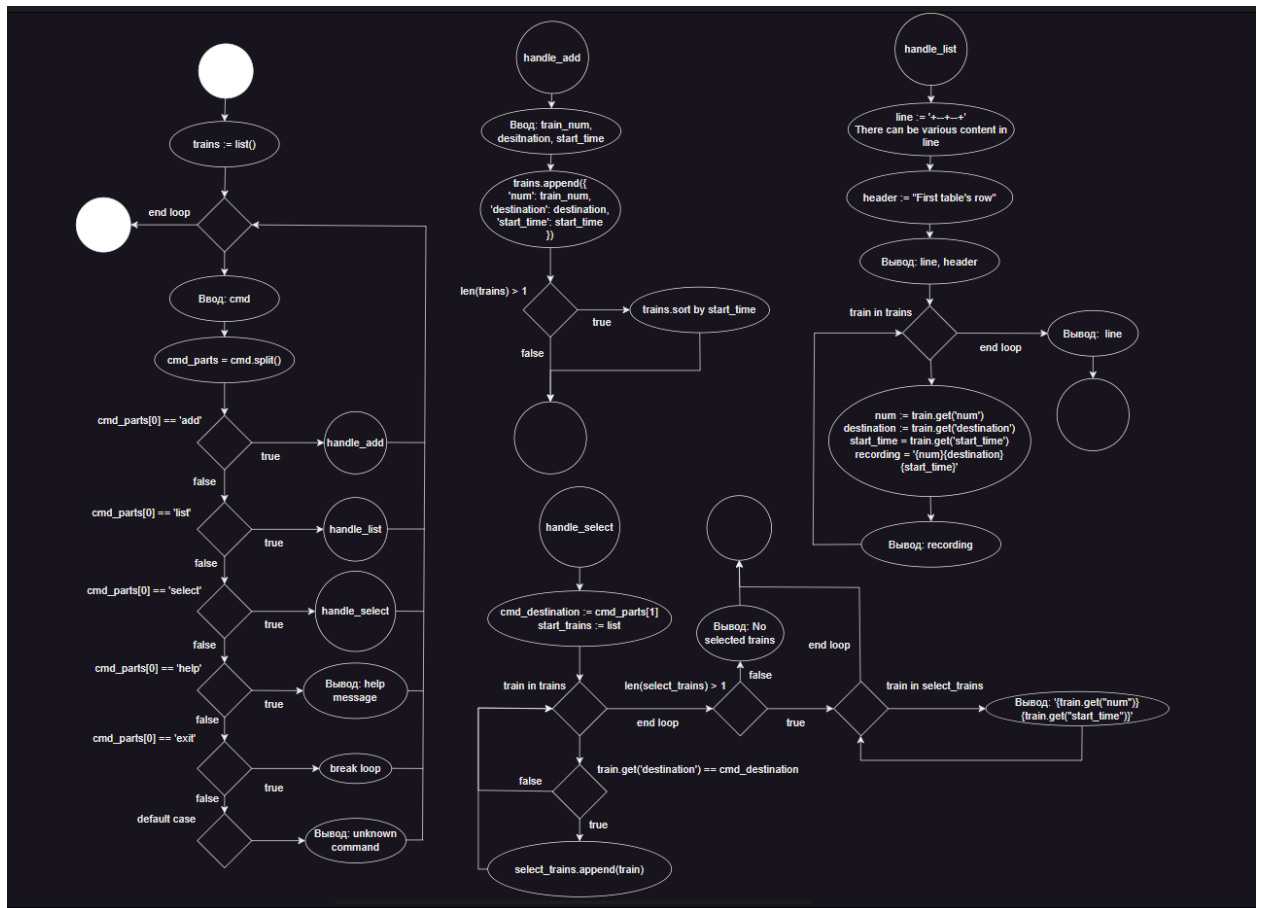


Рисунок 23 – Диаграмма кода индивидуального задания

6. Сошьем ветки и обновим удаленный репозиторий:

```

09d8e26 (HEAD -> develop) individual task
f459583 task2.py is added
22616d4 task1.py is added
9c44581 manual_task1.py is added
5aaf47a (origin/main, origin/HEAD, main) Create README.md
24ed39a Initial commit
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.6\fundamentals_of_software_engineering_lab2_6>

```

Рисунок 24 – Коммиты проекта

```

PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.6\fundamentals_of_software_engineering_lab2_6> git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.6\fundamentals_of_software_engineering_lab2_6> git merge develop
Updating 5aaf47a..09d8e26
Fast-forward
 .gitignore      | 2 +-
 individual_task.py | 52 +++++
 manual_task1.py  | 57 +++++
 task1.py         | 21 +++++
 task2.py         | 9 +++++
 5 files changed, 140 insertions(+), 1 deletion(-)
 create mode 100644 individual_task.py
 create mode 100644 manual_task1.py
 create mode 100644 task1.py
 create mode 100644 task2.py
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.6\fundamentals_of_software_engineering_lab2_6>

```

Рисунок 25 – Объединение веток develop и main

```

PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.6\fundamentals_of_software_engineering_lab2_6> git push origin main
Enumerating objects: 16, done.
Counting objects: 100% (16/16), done.
Delta compression using up to 12 threads
Compressing objects: 100% (14/14), done.
Writing objects: 100% (14/14), 3.38 KiB | 3.38 MiB/s, done.
Total 14 (delta 5), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (5/5), completed with 1 local object.
To https://github.com/PoItaTto/fundamentals_of_software_engineering_lab2_6
 5aaf47a..09d8e26 main -> main
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.6\fundamentals_of_software_engineering_lab2_6>

```

Рисунок 26 – Отправка изменений на удаленный репозиторий

Ответы на вопросы:

1. Что такое словари в языке Python?

Словарь (`dict`) представляет собой структуру данных (которая ещё называется ассоциативный массив), предназначенную для хранения произвольных объектов с доступом по ключу. Данные в словаре хранятся в формате ключ – значение. В языке программирования Python словари (тип `dict`) представляют собой еще одну разновидность структур данных наряду со списками и кортежами. Словарь — это изменяемый (как список) неупорядоченный (в отличие от строк, списков и кортежей) набор элементов "ключ: значение".

2. Может ли функция `len()` быть использована при работе со словарями?

Да, она вернет количество ключей словаря.

3. Какие методы обхода словарей Вам известны?

Существуют несколько методов обхода словарей в Python: использование цикла `for key in dictionary` для перебора ключей, `dictionary.items()` для итерации по парам ключ-значение, `dictionary.keys()` для перебора ключей, и `dictionary.values()` для перебора значений. Также можно воспользоваться методом `dict.fromkeys()`.

4. Какими способами можно получить значения из словаря по ключу?

Используя синтаксис `dict[key]` или методом `dict.get(key)`. Разница в том, что метод вернет `None` или выставленное дефолтное значение, если нет такого ключа, а `dict[key]` выведет ошибку.

5. Какими способами можно установить значение в словаре по ключу?

Через синтаксис `dict[key] = value` или методом `dict.setdefault(key, value)`.

6. Что такое словарь включений?

Словарь включений аналогичен списковым включениям, за исключением того, что он создаёт объект словаря вместо списка.

7. Самостоятельно изучите возможности функции `zip()` приведите примеры ее использования:

Функция `zip()` в Python используется для объединения элементов из различных итерируемых объектов в кортежи. Она создает итератор, который комбинирует элементы из каждого переданного объекта по позициям, формируя кортежи. Пример использования `zip()` через словарь может быть, когда необходимо объединить ключи и значения из двух списков (или других итерируемых объектов) в словарь. Например:

```
keys = ['a', 'b', 'c']
values = [1, 2, 3]

result_dict = dict(zip(keys, values))
# Результат: {'a': 1, 'b': 2, 'c': 3}
```

8. Самостоятельно изучите возможности модуля `datetime`. Каким функционалом по работе с датой и временем обладает этот модуль?

`datetime` в Python предоставляет инструменты для работы с датами и временем. Он позволяет создавать объекты для представления даты, времени или их комбинаций. Модуль позволяет получать текущее время и дату, а также работать с различными форматами даты и времени. В нём реализованы методы для вычислений с датами, такие как вычитание и сложение дней, а также нахождение разницы между двумя датами. Кроме того, он поддерживает изменение временной зоны, форматирование вывода даты и время, и предоставляет возможности для работы с интервалами времени и периодами. Этот модуль является важным инструментом при работе с датами и временем в Python, обеспечивая множество функций для обработки и управления временными данными.