

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 №8
дисциплины «Основы программной инженерии»

Выполнил:
Звездин Алексей Сергеевич
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:
Воронкин Р. А., доцент кафедры
инфокоммуникаций

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2023 г.

Ход работы

1. Я изучил теоретический материал работы

помощью вызова `tuple` генератор преобразуется к кортежу.

Данная задача может быть также решена с помощью списковых включений следующим образом:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import sys

if __name__ == '__main__':
    # Ввести список одной строкой.
    A = list(map(int, input().split()))
    # Проверить количество элементов списка.
    if len(A) != 10:
        print("Неверный размер списка", file=sys.stderr)
        exit(1)

    # Найти искомую сумму.
    s = sum(a for a in A if abs(a) < 5)
    print(s)
```

В этом примере показано использование списковых включений для расчета суммы, однако в отличие от выражения `[a for a in A ...]`, которое на выходе дает нам список, выражение `(a for a in A ...)` дает на выходе специальный объект **генератора**, а не кортеж. Для преобразования генератора в кортеж необходимо воспользоваться вызовом `tuple()`.

Аппаратура и материалы

1. Компьютерный класс общего назначения с конфигурацией ПК не хуже рекомендованной для ОС Windows 10 с подключением к глобальной сети Интернет.

Рисунок 1.1 – Изучение материала для лабораторной работы


2. Создал общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык 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 (*).

Owner *

 InternetHacker1123 ▾

Repository name *

/ lab2(5)_8

✔ Your new repository will be created as lab2-5-8.

The repository name can only contain ASCII letters, digits, and the characters ., -, and _.

Great repository names are short and memorable. Need inspiration? How about **fictional-fiesta** ?

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.](#)

This will set  main as the default branch. Change the default name in your [settings](#).

 You are creating a public repository in your personal account.

Create repository

Рисунок 2.1 – Настройка репозитория

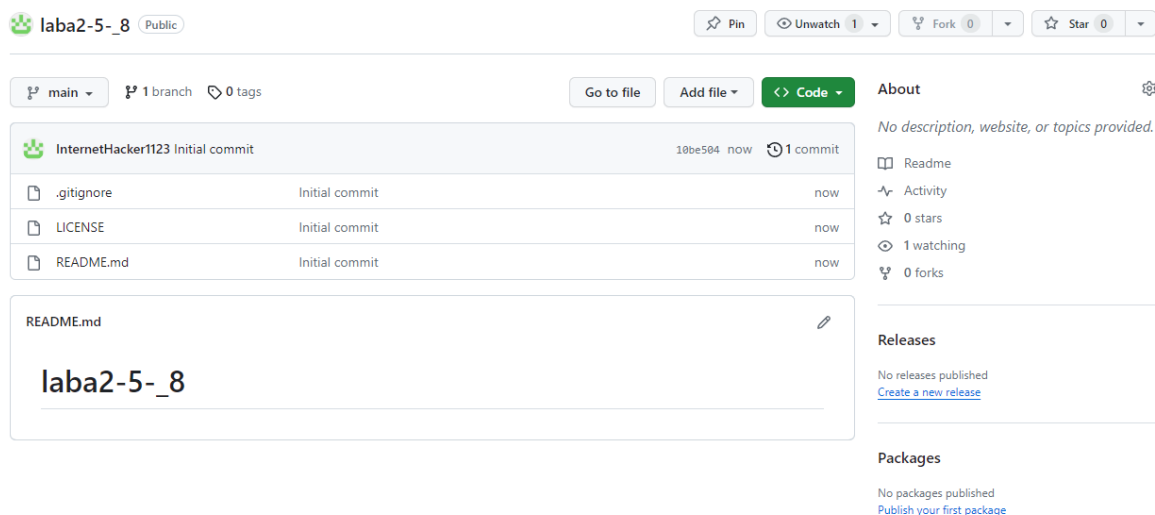


Рисунок 2.2 – Готовый репозиторий

3. Выполняю клонирование созданного репозитория

```
C:\Windows\system32\cmd.exe

C:\Users\tyt\Desktop\SE\laba8>git clone https://github.com/InternetHacker1123/laba2-5-8.git
Cloning into 'laba2-5-8'...
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.

C:\Users\tyt\Desktop\SE\laba8>
```

Рисунок 3.1 – Клонирование репозитория на локальный диск

4. Дополнил файл .gitignore необходимыми правилами для работы с VS Code

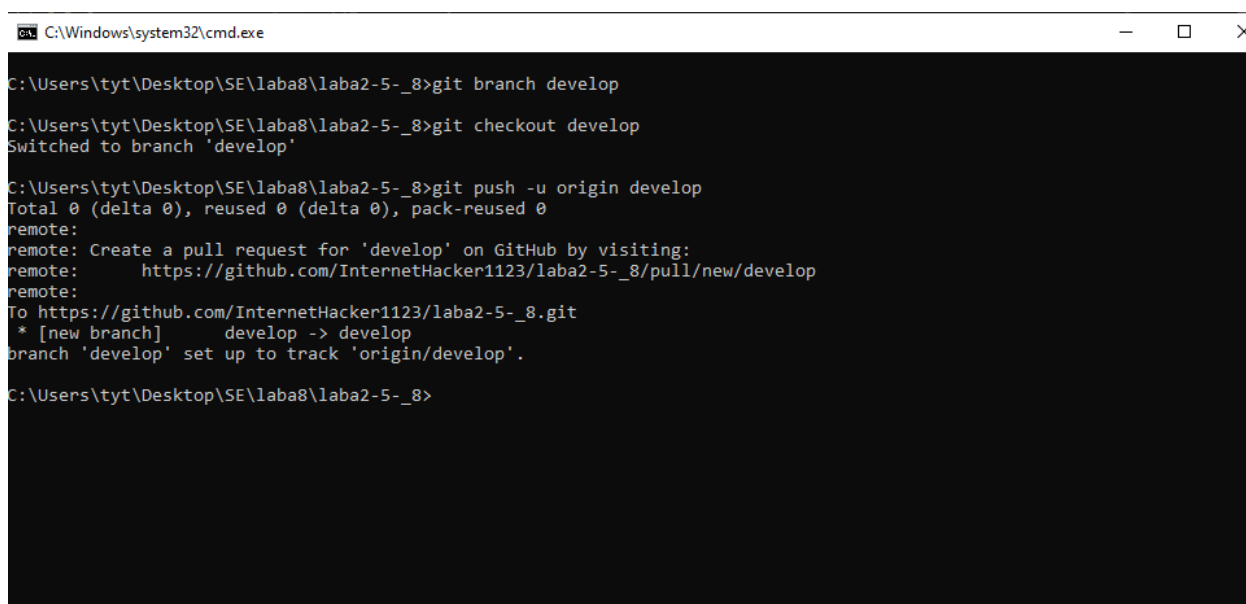
```

155  .vscode/*
156  !.vscode/settings.json
157  !.vscode/tasks.json
158  !.vscode/launch.json
159  !.vscode/extensions.json
160  !.vscode/*.code-snippets
161
162  # Local History for Visual Studio Code
163  .history/
164
165  # Built Visual Studio Code Extensions
166  *.vsix
167

```

Рисунок 4.1 – .gitignore для VS Code

5. Организовал свой репозиторий в соответствии с моделью ветвления git-flow



```

C:\Windows\system32\cmd.exe
C:\Users\tyt\Desktop\SE\laba8\laba2-5-_8>git branch develop
C:\Users\tyt\Desktop\SE\laba8\laba2-5-_8>git checkout develop
Switched to branch 'develop'
C:\Users\tyt\Desktop\SE\laba8\laba2-5-_8>git push -u origin develop
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'develop' on GitHub by visiting:
remote:   https://github.com/InternetHacker1123/laba2-5-_8/pull/new/develop
remote:
To https://github.com/InternetHacker1123/laba2-5-_8.git
 * [new branch]      develop -> develop
branch 'develop' set up to track 'origin/develop'.
C:\Users\tyt\Desktop\SE\laba8\laba2-5-_8>

```

Рисунок 5.1 – Создание ветки develop от ветки main

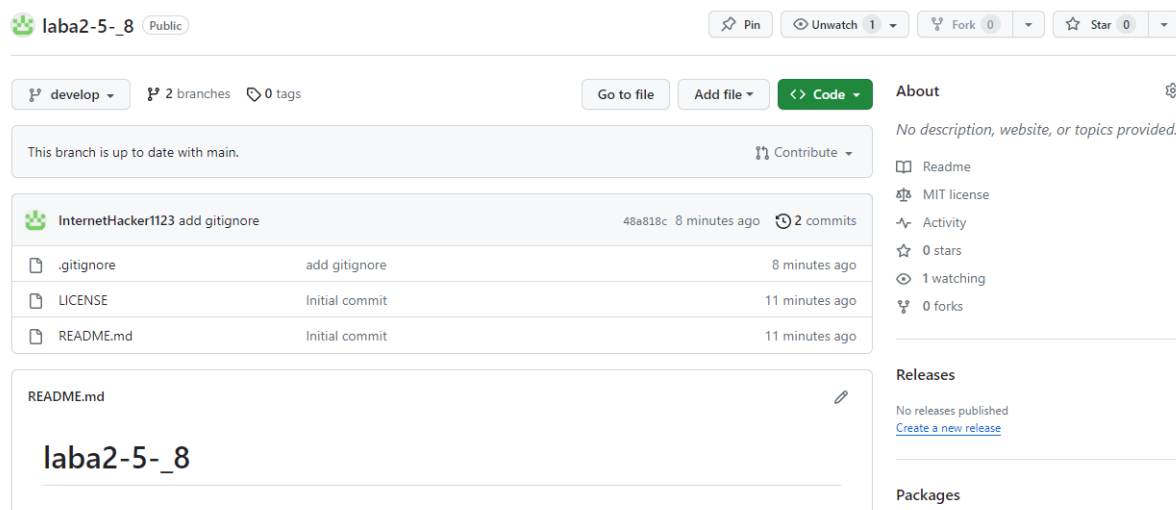


Рисунок 5.2 – Ветка develop на GitHub

6. Создал проект PyCharm в папке репозитория

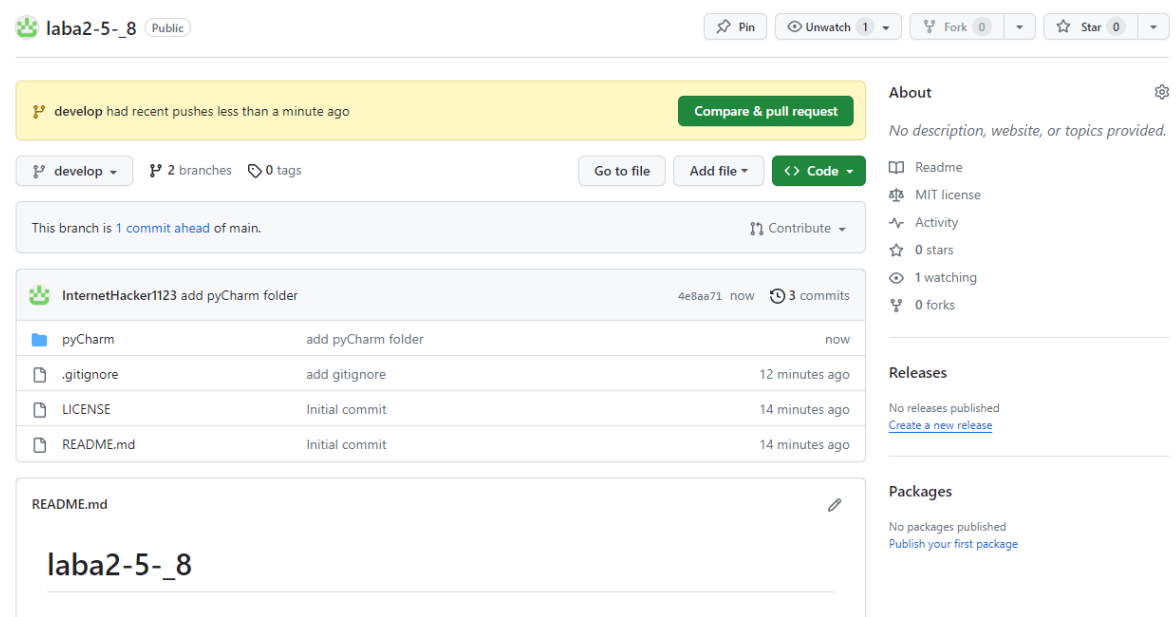


Рисунок 6.1 – Репозиторий с проектом PyCharm

7. Проработал примеры лабораторной работы. Создал для каждого примера отдельный модуль языка Python. Зафиксировал изменения в репозитории.

```
laba2-5-8 > pyCharm > example1_1.py > ...
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  import sys
5
6  if __name__ == '__main__':
7      # Ввести кортеж одной строкой.
8      A = tuple(map(int, input().split()))
9      # Проверить количество элементов кортежа.
10     if len(A) != 10:
11         print("Неверный размер кортежа", file=sys.stderr)
12         exit(1)
13
14     # Найти искомую сумму.
15     s = 0
16     for item in A:
17         if abs(item) < 5:
18             s += item
19
20     print(s)
```

Рисунок 7.1 – Проработка примера 1 с применением map()

```
laba2-5-8 > pyCharm > example1_0.py > ...
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  import sys
5
6  if __name__ == '__main__':
7      # Ввести список одной строкой.
8      A = list(map(int, input().split()))
9      # Проверить количество элементов списка.
10     if len(A) != 10:
11         print("Неверный размер списка", file=sys.stderr)
12         exit(1)
13
14     # Найти искомую сумму.
15     s = sum(a for a in A if abs(a) < 5)
16     print(s)
```

Рисунок 7.2 – Проработка примера 1 с применением списковых включений

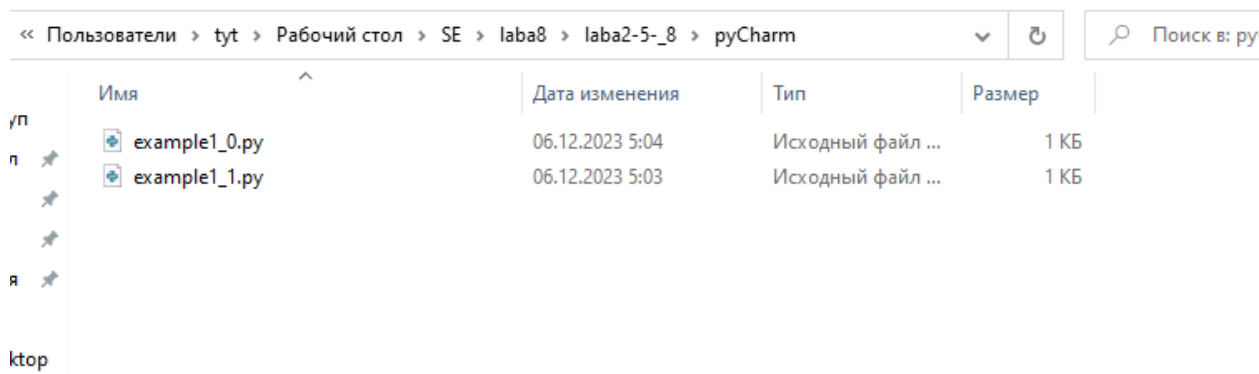


Рисунок 7.4 – Создание отдельных модулей для каждого из примеров



Рисунок 7.5 – Фиксирование изменений в репозитории

8. Привел в отчете скриншоты результатов выполнения каждой из программ примеров при различных исходных данных, вводимых с клавиатуры.



Рисунок 8.1 – Результат примера 1 с применением map()

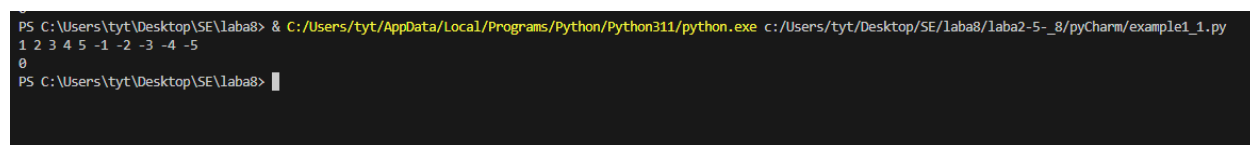


Рисунок 8.2 – Результат примера 2 с применением списковых включений

9. Привел в отчете скриншоты работы программ решения индивидуальных заданий


```
laba2-5-8 > pyCharm > individual1.py > ...
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  import sys
5
6  if __name__ == '__main__':
7      A = tuple(map(int, input().split()))
8      if len(A) < 3:
9          print("Слишком маленькая длина кортежа", file=sys.stderr)
10         exit(1)
11
12         for i in range(len(A) - 2):
13             if A[i+1] > A[i] and A[i+1] > A[i+2]:
14                 print(i, i+1, i+2)
15                 break
16         else:
17             print("Ничего не нашлось")
```

Рисунок 9.1 – Код программы individual1.py

10. Зафиксировал сделанные изменения в репозитории

```
C:\Windows\system32\cmd.exe
C:\Users\tyt\Desktop\SE\laba8\laba2-5-8>git add .
C:\Users\tyt\Desktop\SE\laba8\laba2-5-8>git commit -m"add examples"
[develop 226fc54] add examples
 2 files changed, 36 insertions(+)
C:\Users\tyt\Desktop\SE\laba8\laba2-5-8>git add .
C:\Users\tyt\Desktop\SE\laba8\laba2-5-8>git commit -m"final"
[develop bcb4e4c] final
 1 file changed, 17 insertions(+)
 create mode 100644 pyCharm/individual1.py
C:\Users\tyt\Desktop\SE\laba8\laba2-5-8>
```

Рисунок 10.1 – Коммит файлов в репозитории git

Контрольные вопросы

1. Что такое кортежи в языке Python?

Кортеж (tuple) – это неизменяемая структура данных, которая по своему подобию очень похожа на список. С кортежем мы не можем производить такие операции, т. к. элементы его изменять нельзя.

2. Каково назначение кортежей в языке Python?

Существует несколько причин, по которым стоит использовать кортежи вместо списков. Одна из них – это обезопасить данные от случайного изменения. Если мы получили откуда-то массив данных, и у нас есть желание поработать с ним, но при этом непосредственно менять данные мы не собираемся, тогда, это как раз тот случай, когда кортежи придутся как нельзя кстати. Используя их в данной задаче, мы дополнительно получаем сразу несколько бонусов – во-первых, это экономия места. Дело в том, что кортежи в памяти занимают меньший объем по сравнению со списками.

Во-вторых – прирост производительности, который связан с тем, что кортежи работают быстрее, чем списки (т. е. на операции перебора элементов и т. п. будет тратиться меньше времени). Важно также отметить, что кортежи можно использовать в качестве ключа у словаря.

3. Как осуществляется создание кортежей?

Кортеж с заданным содержанием создается так же, как список, только вместо квадратных скобок используются круглые. При желании можно воспользоваться функцией tuple().

4. Как осуществляется доступ к элементам кортежа?

Доступ к элементам кортежа осуществляется так же, как к элементам списка – через указание индекса.

5. Зачем нужна распаковка (деструктуризация) кортежа?

Обращение по индексу, это не самый удобный способ работы с кортежами. Дело в том, что кортежи часто содержат значения разных типов, и помнить, по какому индексу что лежит — очень непросто. В этом случае можно воспользоваться деструктуризацией.

6. Какую роль играют кортежи в множественном присваивании?

Благодаря тому, что кортежи легко собирать и разбирать, в Python удобно делать такие вещи, как множественное присваивание. Используя множественное присваивание, можно провернуть интересный трюк: обмен значениями между двумя переменными.

7. Как выбрать элементы кортежа с помощью среза?

С помощью операции взятия среза можно получить другой кортеж. Общая форма операции взятия среза для кортежа, следующая:

```
T2 = T1[i:j]
```

8. Как выполняется конкатенация и повторение кортежей?

Для кортежей можно выполнять операцию конкатенации, которая обозначается символом `+`. Кортеж может быть образован путем операции повторения, обозначаемой символом `*`.

9. Как выполняется обход элементов кортежа?

Элементы кортежа можно последовательно просмотреть с помощью операторов цикла `while` или `for`.

10. Как проверить принадлежность элемента кортежу?

Проверить принадлежность элемента кортежу можно с помощью операции `in`.

11. Какие методы работы с кортежами Вам известны?

Чтобы получить индекс (позицию) элемента в кортеже, нужно использовать метод `index()`. Чтобы определить количество вхождений заданного элемента в кортеж используется метод `count`

12. Допустимо ли использование функций агрегации таких как `len()` , `sum()` и т. д. при работе с кортежами?

Да, можно использовать функции агрегации, такие как `len()`, `sum()` и другие, при работе с кортежами в Python.

13. Как создать кортеж с помощью спискового включения.

```
my_tuple = tuple(expression for item in iterable)
```