

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

Кафедра инфокоммуникаций

**Языки программирования
Отчет по лабораторной работе № 2
«Работа с кортежами в языке Python».**

Выполнил студент группы

ИТС-б-о-20-1 (1)

Рудаков В.Б. « » _____ 20__ г.

Подпись студента _____

Работа защищена « » _____ 20__ г.

Проверил доцент

Кафедры инфокоммуникаций, кандидат
технических наук

Воронкин Р.А.

(подпись)

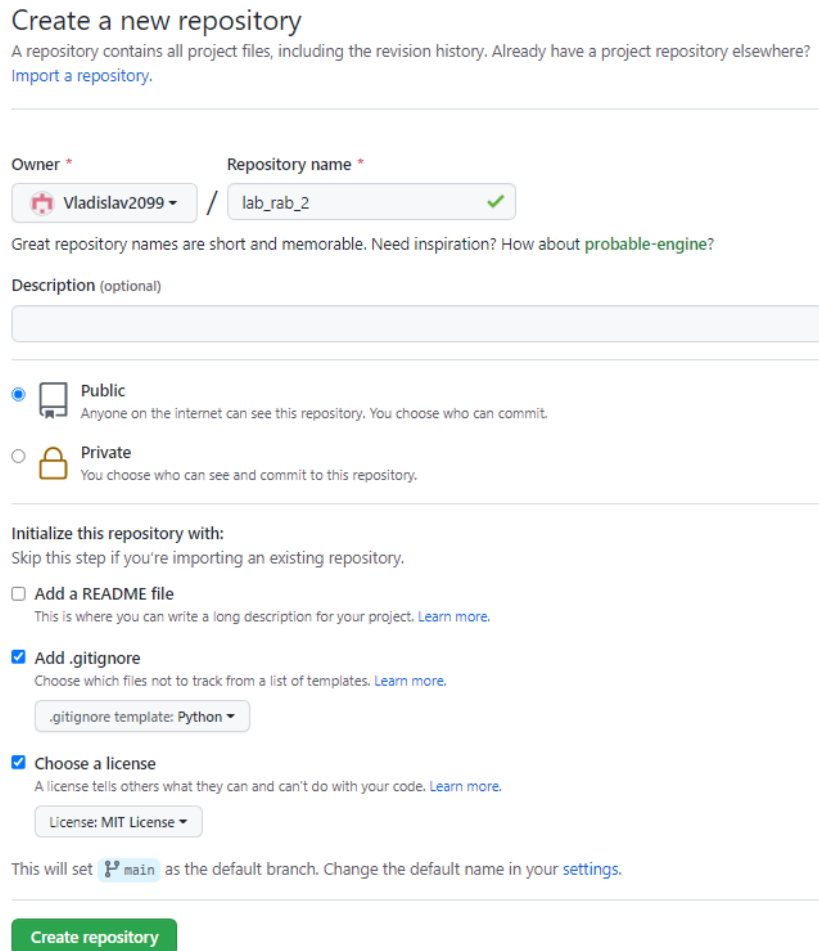
Ставрополь, 2021

Тема работы: Работа с кортежами в языке Python.

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

Ссылка на репозиторий: https://github.com/Vladislav2099/lab_rab_2/tree/main

1. Создать 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.](#)

Owner * Vladislav2099 / Repository name * lab_rab_2 ✓

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

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](#).

[Create repository](#)

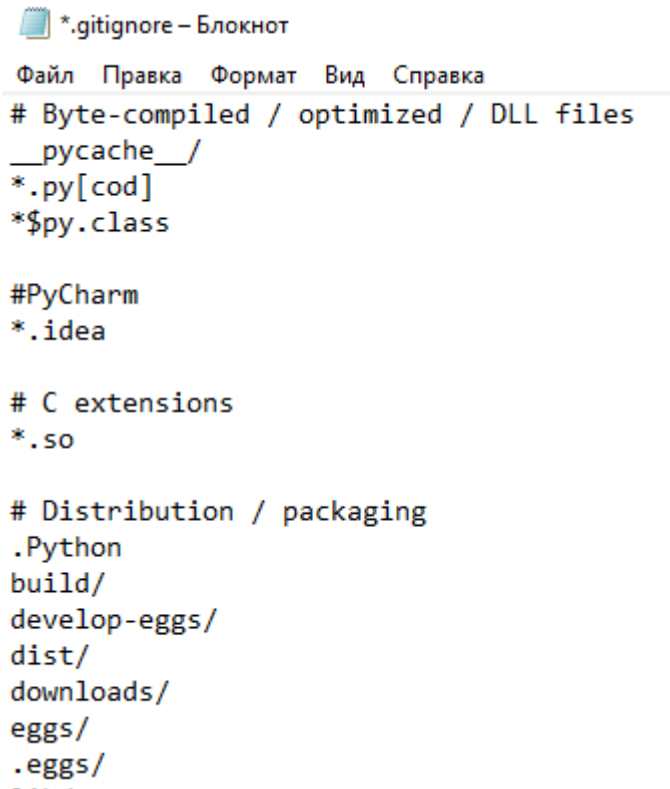
Рисунок 1. Создание репозитория.

2. Выполните клонирование созданного репозитория.

```
User@DESKTOP-ЮОСМНМТ MINGW64 /j/Учёба/Языки Программирования/lab_rab_2
$ git clone https://github.com/Vladislav2099/lab_rab_2.git
Cloning into 'lab_rab_2'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (4/4), done.
```

Рисунок 2. Клонирование репозитория.

3. Дополните файл `.gitignore` необходимыми правилами для работы с IDE PyCharm.



The screenshot shows a Notepad window titled `*.gitignore – Блокнот`. The menu bar includes `Файл`, `Правка`, `Формат`, `Вид`, and `Справка`. The content of the `.gitignore` file is as follows:

```
# Byte-compiled / optimized / DLL files
__pycache__/
*.py[cod]
*$py.class

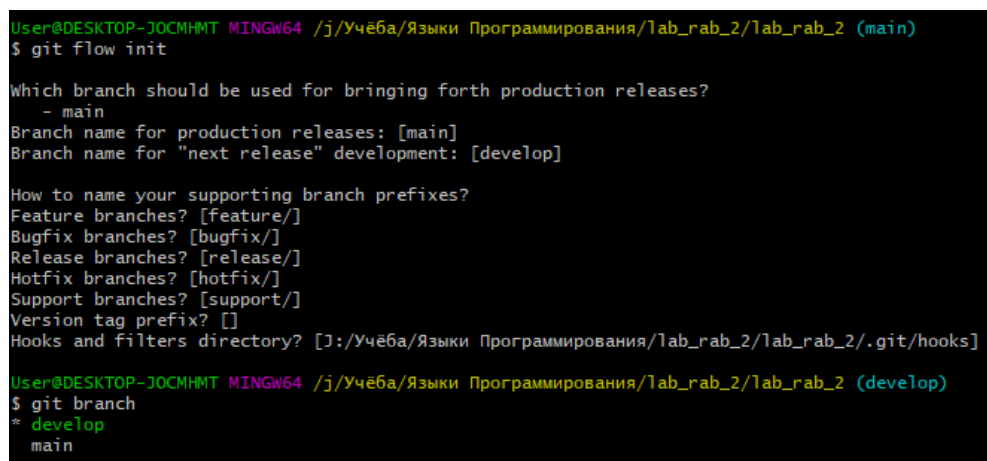
#PyCharm
*.idea

# C extensions
*.so

# Distribution / packaging
.Python
build/
develop-eggs/
dist/
downloads/
eggs/
.eggs/
...
```

Рисунок 3. Редактирование файла `.gitignore`.

4. Организуйте свой репозиторий в соответствии с моделью ветвления `git-flow`.



The screenshot shows a terminal window with the following commands and output:

```
User@DESKTOP-JOCMHMT MINGW64 /j/Учеба/Языки Программирования/lab_rab_2/lab_rab_2 (main)
$ git flow init

Which branch should be used for bringing forth production releases?
- main
Branch name for production releases: [main]
Branch name for "next release" development: [develop]

How to name your supporting branch prefixes?
Feature branches? [feature/]
Bugfix branches? [bugfix/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? []
Hooks and filters directory? [j:/Учеба/Языки Программирования/lab_rab_2/lab_rab_2/.git/hooks]

User@DESKTOP-JOCMHMT MINGW64 /j/Учеба/Языки Программирования/lab_rab_2/lab_rab_2 (develop)
$ git branch
* develop
main
```

Рисунок 4. Использование ветвления `git-flow`.

5. Создайте проект PyCharm в папке репозитория.

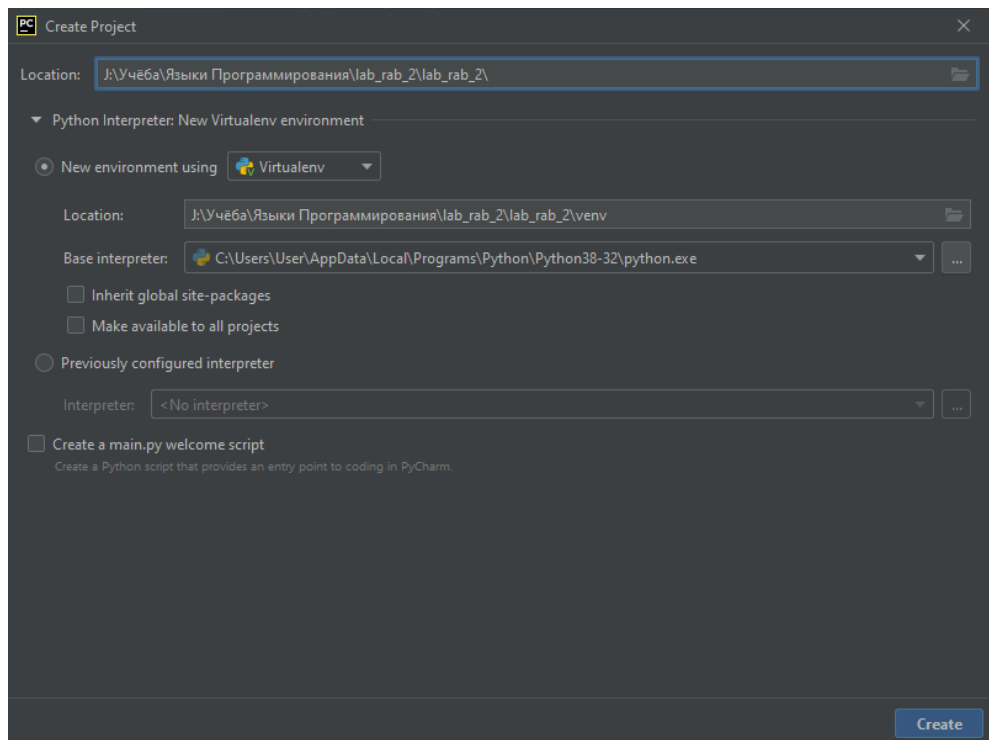
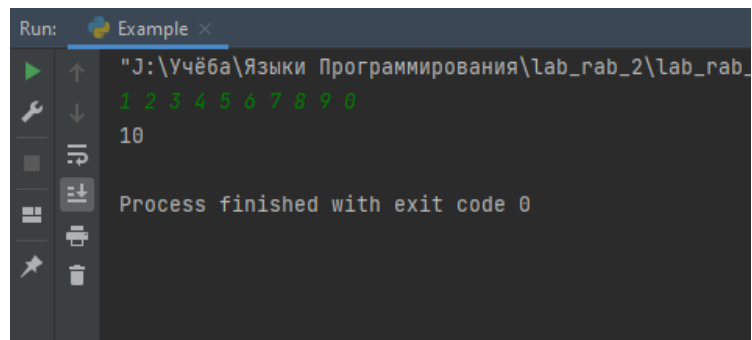


Рисунок 5. Создание проекта PyCharm.

6. Проработайте пример лабораторной работы. Создайте для него отдельный модуль языка Python. Зафиксируйте изменения в репозитории.

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



```
Run: Example x
"J:\Учёба\Языки Программирования\lab_rab_2\lab_rab_2.py"
1 2 3 4 5 6 7 8 9 0
10
Process finished with exit code 0
```

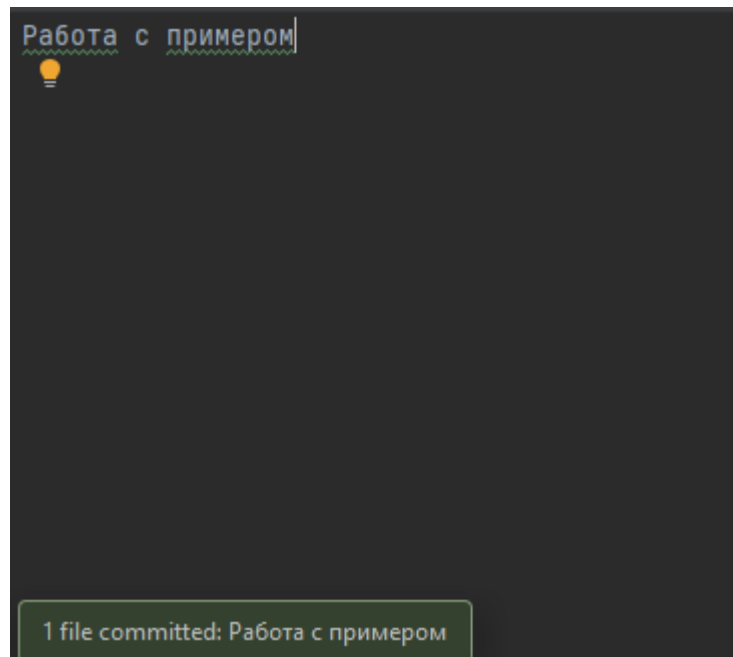
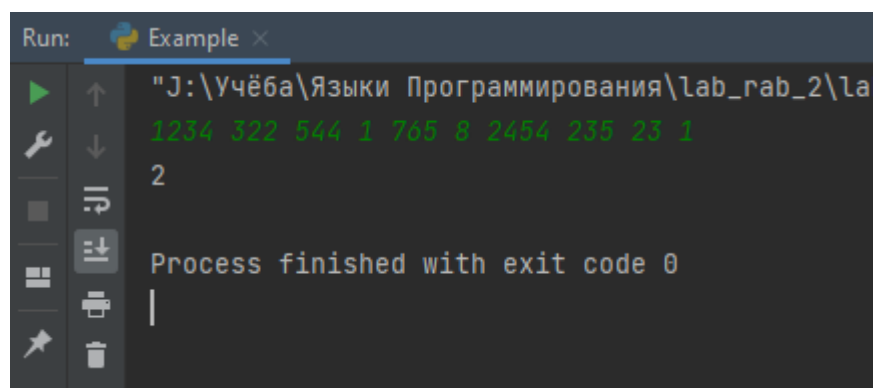


Рисунок 6. Пример лабораторной работы.

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



```
Run: Example x
"J:\Учёба\Языки Программирования\lab_rab_2\lab_rab_2.py"
1234 322 544 1 765 8 2454 235 23 1
2
Process finished with exit code 0
```

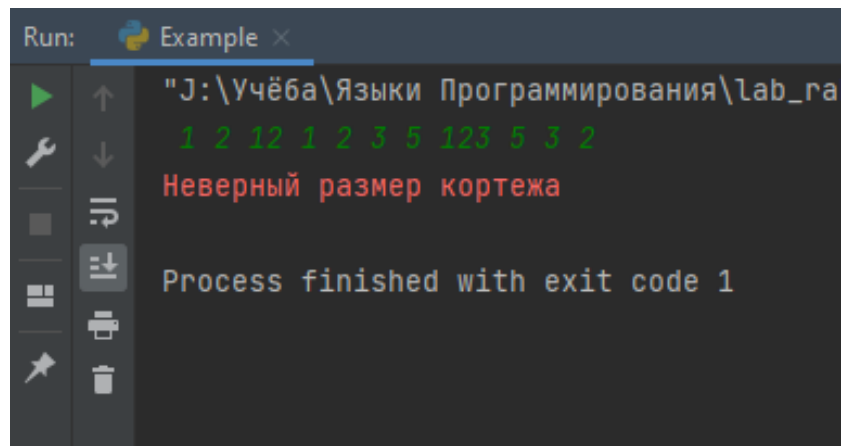


Рисунок 7. Пример с различными исходными данными.

8. Выполните слияние ветки для разработки с веткой main.

```
User@DESKTOP-JOCMHMT MINGW64 /j/Учёба/Языки Программирования/lab_rab_2/lab_rab_2 (develop)
$ git checkout main
Switched to branch 'main'
Your branch is ahead of 'origin/main' by 1 commit.
(use "git push" to publish your local commits)

User@DESKTOP-JOCMHMT MINGW64 /j/Учёба/Языки Программирования/lab_rab_2/lab_rab_2 (main)
$ git merge develop
Updating 2bc00a7..577bfd0
Fast-forward
 Example.py | 18 ++++++
 1 file changed, 18 insertions(+)
 create mode 100644 Example.py
```

Рисунок 8. Слияние веток.

9. Отправьте сделанные изменения на сервер GitHub.

```
User@DESKTOP-JOCMHMT MINGW64 /j/Учёба/Языки Программирования/lab_rab_2/lab_rab_2 (main)
$ git push origin main
Enumerating objects: 8, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 2 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 1010 bytes | 202.00 KiB/s, done.
Total 6 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Vladislav2099/lab_rab_2.git
 1486b9a..577bfd0 main -> main
```

Рисунок 9. Отправка изменений.

КОНТРОЛЬНЫЕ ВОПРОСЫ:

1. Что такое списки в языке Python?

Список – это нумерованный набор объектов.

2. Каково назначение кортежей в языке Python?

Одна из причин – это безопасность данные от случайного изменения.

3. Как осуществляется создание кортежей?

```
>>> a = ()
```

```
>>> print(type(a))
```

```
>>> a = tuple([1, 2, 3, 4])
```

```
>>> print(a)
```

4. Как осуществляется доступ к элементам кортежа?

Доступ к элементам кортежа осуществляется также как к элементам списка – через указание индекса

5. Зачем нужна распаковка (деструктуризация) кортежа?

Для упрощённого обращения к данным кортежа, не только используя индекс.

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

```
(a, b, c) = (1, 2, 3)
```

```
a # 1
```

```
b # 2
```

```
c # 3
```

7. Как выбрать элементы кортежа с помощью среза?

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

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

здесь

T2 – новый кортеж, который получается из кортежа T1;

T1 – исходный кортеж, для которого происходит срез;

i, j – соответственно нижняя и верхняя границы среза. Фактически берутся ко вниманию элементы, лежащие на позициях i, i+1, ..., j-1. Значение j определяет позицию за последним элементом среза.

Операция взятия среза для кортежа может иметь модификации такие же как и для списков.

8. Как выполняется конкатенация и повторение кортежей?

Для кортежей можно выполнять операцию конкатенации, которая обозначается символом $+$.

$$T3 = T1 + T2$$

где

$T1, T2$ – кортежи, для которых нужно выполнить операцию конкатенации. Операнды $T1, T2$ обязательно должны быть кортежами. При выполнении операции конкатенации для кортежей, использовать в качестве операндов любые другие типы (строки, списки) запрещено;

$T3$ – кортеж, который есть результатом.

9. Как выполняется обход элементов кортежа?

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

10. Как проверить принадлежность элемента кортежу?

Проверка вхождения элемента в кортеж

Оператор in

Заданный кортеж, который содержит строки

$A = ("abc", "abcd", "bcd", "cde")$

Ввести элемент

$item = str(input("s = "))$

if (item in A):

print(item, " in ", A, " = True")

else:

print(item, " in ", A, " = False")

11. Какие методы работы с кортежами Вам известны?

Метод `index()`. Поиск позиции элемента в кортеже:

$pos = T.index(item)$

Метод `count()`. Количество вхождений элемента в кортеж:

$k = T.count(item)$

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

Допустимо, если условия использования той или иной функции не нарушены.

13. Как создать кортеж с помощью спискового включения.

(a for a in A ...) дает на выходе специальный объект генератора, а не кортеж. Для преобразования генератора в кортеж необходимо воспользоваться вызовом *tuple()*.