

Лабораторная работа 2.5

Тема: 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.

Порядок выполнения работы


1. Создаём аккаунт в GitHub. Затем создаём новый общедоступный репозиторий, 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 *

 RomanGorchakov ▾

Repository name *

/ Test

✓ Test is available.

Great repository names are short and memorable. Need inspiration? How about [automatic-octo-chainsaw](#) ?

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.](#)

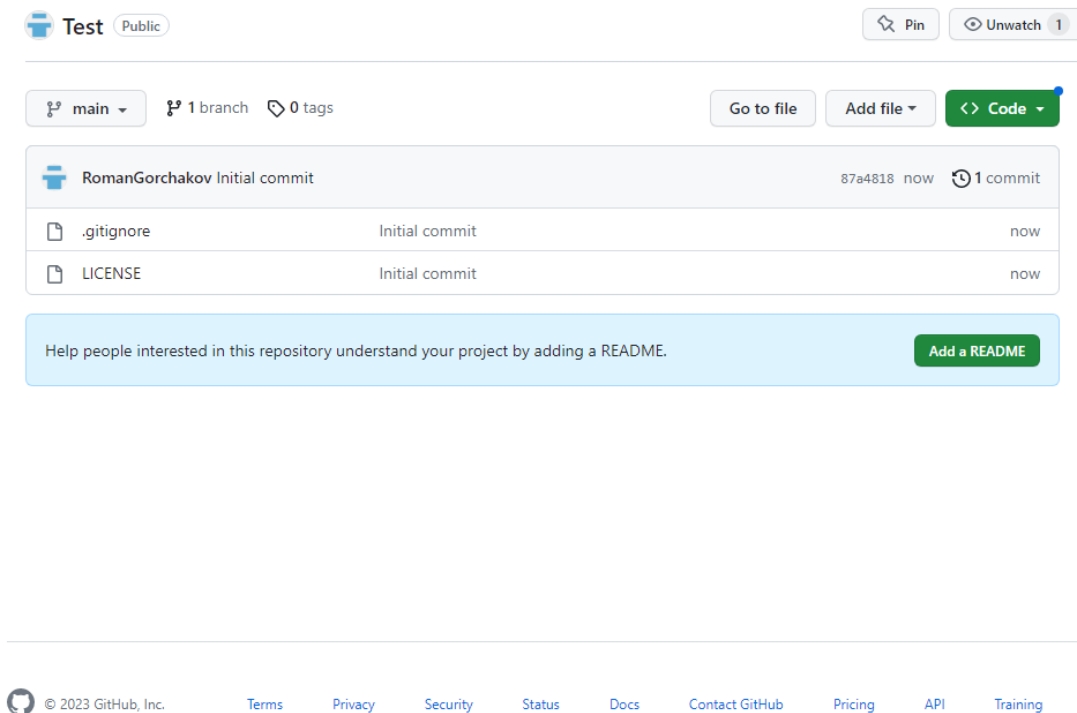
Choose a license

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



2. Теперь необходимо дополнить файл `.gitignore` с необходимыми правилами для языка программирования Python. Для этого переходим по ссылке «<https://github.com/github/gitignore>» и скачиваем оттуда файл «Python.gitignore».

	Phalcon.gitignore	Remove trailing asterisks in Phalcon rules	10 years ago
	PlayFramework.gitignore	Added /project/project to PlayFramework.gitignore	7 years ago
	Plone.gitignore	Covered by global vim template	10 years ago
	Prestashop.gitignore	Update for Prestashop 1.7 (#3261)	3 years ago
	Processing.gitignore	Ignore transpiled .java and .class files (#3016)	4 years ago
	PureScript.gitignore	Update PureScript adding .spago (#3278)	3 years ago
	Python.gitignore	Update Python.gitignore	last year
	Qooxdoo.gitignore	Add gitignore for qooxdoo apps	13 years ago
	Qt.gitignore	Remove trailing whitespace	2 years ago
	R.gitignore	Merge pull request #3792 from jl5000/patch-1	2 years ago
	README.md	Merge pull request #3854 from AnilSeervi/patch-1	2 years ago
	ROS.gitignore	Added ignore for files created by <code>catkin_make_isolated</code>	6 years ago
	Racket.gitignore	Update Racket.gitignore	2 years ago
	Rails.gitignore	Ignore Rails .env according recommendations	2 years ago
	Raku.gitignore	Changes the name of Perl 6 to Raku (#3312)	3 years ago
	RhodesRhomobile.gitignore	Add Rhodes mobile application framework gitignore	13 years ago
	Ruby.gitignore	Ruby: ignore RuboCop remote inherited config files (#3197)	4 years ago

```
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/
24 *.egg-info/
25 .installed.cfg
26 *.egg
27 MANIFEST
28
```

3. Теперь создаём файл «README.md», где вносим информацию о своей группе и ФИО. Сохраняем набранный текст через кнопку «Commit changes».



4. После этого нужно организовать репозиторий в соответствии с моделью ветвления Git-flow. Для этого В окне «Codespace» выбираем опцию «Create codespace on main», где введём команды: «git branch develop» и «git push -u origin develop» для создания ветки разработки; «git branch feature_branch» для создания ветки функций; «git branch release/1.0.0» для создания ветки релиза; «git checkout main» и «git branch hotfix» для создания веток hotfix.

```
• @RomanGorchakov → /workspaces/Py5 (main) $ git checkout -b develop
  Switched to a new branch 'develop'
• @RomanGorchakov → /workspaces/Py5 (develop) $ git checkout -b feature_branch
  Switched to a new branch 'feature_branch'
• @RomanGorchakov → /workspaces/Py5 (feature_branch) $ git checkout develop
  Switched to branch 'develop'
• @RomanGorchakov → /workspaces/Py5 (develop) $ git checkout -b release/1.0.0
  Switched to a new branch 'release/1.0.0'
• @RomanGorchakov → /workspaces/Py5 (release/1.0.0) $ git checkout main
  Switched to branch 'main'
  Your branch is up to date with 'origin/main'.
• @RomanGorchakov → /workspaces/Py5 (main) $ git checkout -b hotfix
  Switched to a new branch 'hotfix'
• @RomanGorchakov → /workspaces/Py5 (hotfix) $ git checkout develop
  Switched to branch 'develop'
○ @RomanGorchakov → /workspaces/Py5 (develop) $
```

5. Создаём файл «example.py», в котором пользователю нужно ввести кортеж из 10 элементов, и программа должна найти сумму элементов, по модулю меньших 5, и вывести её на экран.

```
6 8 -3 2 3 0 -3 -6 -9 0
-1

...Program finished with exit code 0
Press ENTER to exit console. □
```

6. Создаём файл «individual.py», в котором пользователю нужно ввести кортеж, а программа должна найти хотя бы одну пару одинаковых соседних элементов и вывести номера элементов первой из таких пар при положительном результате.

```
6 8 -3 -3 2 3 0 -6 -9
2 3

...Program finished with exit code 0
Press ENTER to exit console. □
```

7. Выполняем коммит файлов в репозиторий Git в ветку разработки, сливаем её с веткой main и отправляем изменения на сервер GitHub.

```

● @RomanGorchakov → /workspaces/Py5 (develop) $ git add .
● @RomanGorchakov → /workspaces/Py5 (develop) $ git commit -m "Python files"
[develop 4bef1eb] Python files
2 files changed, 30 insertions(+)
create mode 100644 example.py
create mode 100644 individual.py
● @RomanGorchakov → /workspaces/Py5 (develop) $ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
● @RomanGorchakov → /workspaces/Py5 (main) $ git merge develop
Updating 3eef270..4bef1eb
Fast-forward
 example.py | 18 +++++
 individual.py | 12 +++++
2 files changed, 30 insertions(+)
create mode 100644 example.py
create mode 100644 individual.py
● @RomanGorchakov → /workspaces/Py5 (main) $ git push -u
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 2 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 836 bytes | 836.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/RomanGorchakov/Py5
3eef270..4bef1eb main -> main
branch 'main' set up to track 'origin/main'.
○ @RomanGorchakov → /workspaces/Py5 (main) $ 

```

Контрольные вопросы

1. Что такое списки в языке Python?

Список (list) – это структура данных для хранения объектов различных типов.

Кортеж (tuple) – неизменяемая структура данных, которая по своему подобию очень похожа на список.

2. Каково назначение кортежей в языке Python?

Существует несколько причин, по которым стоит использовать кортежи вместо списков. Одна из них – 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).

3. Как осуществляется создание кортежей?

Для создания пустого кортежа можно воспользоваться функцией `tuple()`.

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?

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. Но есть способ лучше! Как мы кортеж собираем, так его можно и разобрать. Именно таким способом принято получать и сразу разбирать значения, которые возвращает функция.

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.

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?

С помощью оператора `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, допустимо.

13. Как создать кортеж с помощью спискового включения.

`T1 = (i for i in range()).`