

Лабораторная работа 2.4

Тема: Работа со списками в языке Python.

Цель работы: приобретение навыков по работе со списками 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.](#)

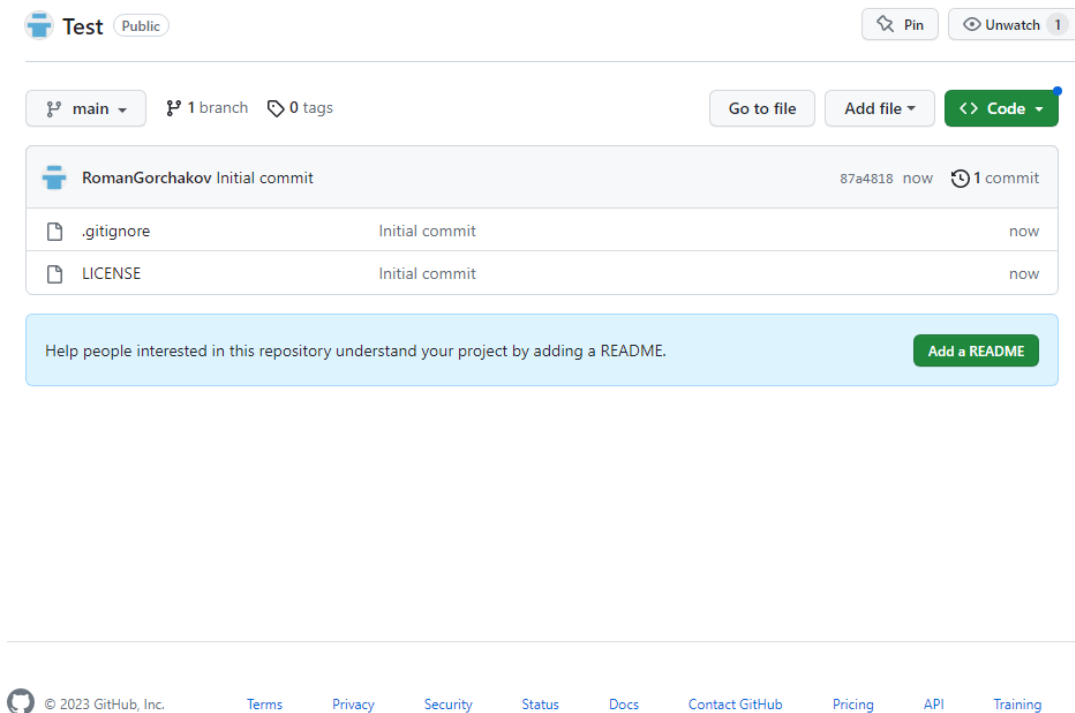
Choose a license

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
```

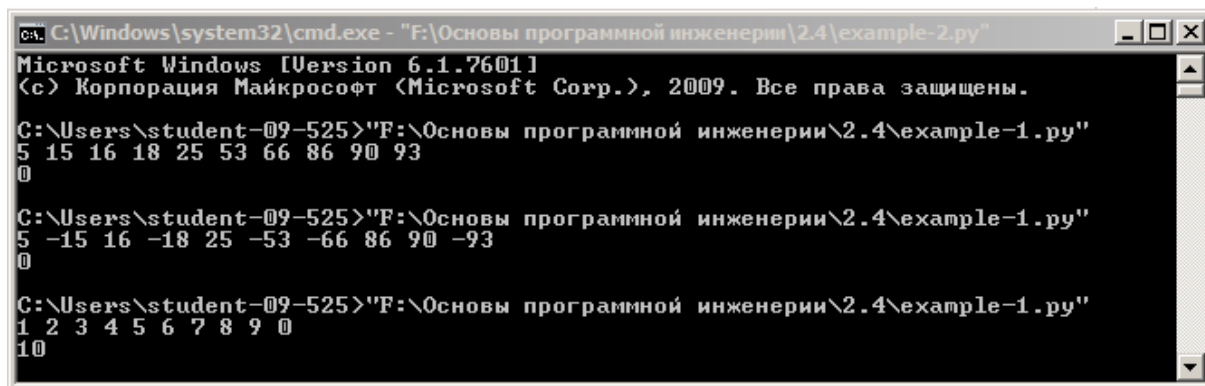
3. Теперь создаём файл «README.md», где вносим информацию о своей группе и ФИО. Сохраняем набранный текст через кнопку «Commit changes».



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/Py3 (main) $ git checkout -b develop
Switched to a new branch 'develop'
● @RomanGorchakov →/workspaces/Py3 (develop) $ git checkout -b feature_branch
Switched to a new branch 'feature_branch'
● @RomanGorchakov →/workspaces/Py3 (feature_branch) $ git checkout develop
Switched to branch 'develop'
● @RomanGorchakov →/workspaces/Py3 (develop) $ git checkout -b release/1.0.0
Switched to a new branch 'release/1.0.0'
● @RomanGorchakov →/workspaces/Py3 (release/1.0.0) $ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
● @RomanGorchakov →/workspaces/Py3 (main) $ git checkout -b hotfix
Switched to a new branch 'hotfix'
● @RomanGorchakov →/workspaces/Py3 (hotfix) $ git checkout develop
Switched to branch 'develop'
● @RomanGorchakov →/workspaces/Py3 (develop) $ git branch
* develop
  feature_branch
  hotfix
  main
  release/1.0.0
○ @RomanGorchakov →/workspaces/Py3 (develop) $
```

5. Создаём файл «example-1.py», в котором пользователю нужно ввести список из 10 элементов, и программа должна найти сумму элементов, меньших 5 по модулю, и вывести её на экран.



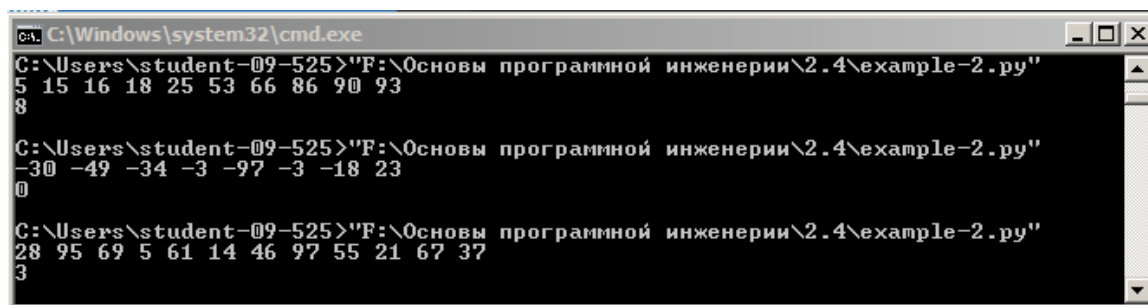
```
C:\Windows\system32\cmd.exe - "F:\Основы программной инженерии\2.4\example-2.py"
Microsoft Windows [Version 6.1.7601]
(c) Корпорация Майкрософт (Microsoft Corp.), 2009. Все права защищены.

C:\Users\student-09-525>"F:\Основы программной инженерии\2.4\example-1.py"
5 15 16 18 25 53 66 86 90 93
0

C:\Users\student-09-525>"F:\Основы программной инженерии\2.4\example-1.py"
5 -15 16 -18 25 -53 -66 86 90 -93
0

C:\Users\student-09-525>"F:\Основы программной инженерии\2.4\example-1.py"
1 2 3 4 5 6 7 8 9 0
10
```

6. Создаём файл «example-2.py», в котором пользователю нужно ввести целочисленный список, и программа должна определить количество элементов, располагающихся между минимальным и максимальным элементами списка.

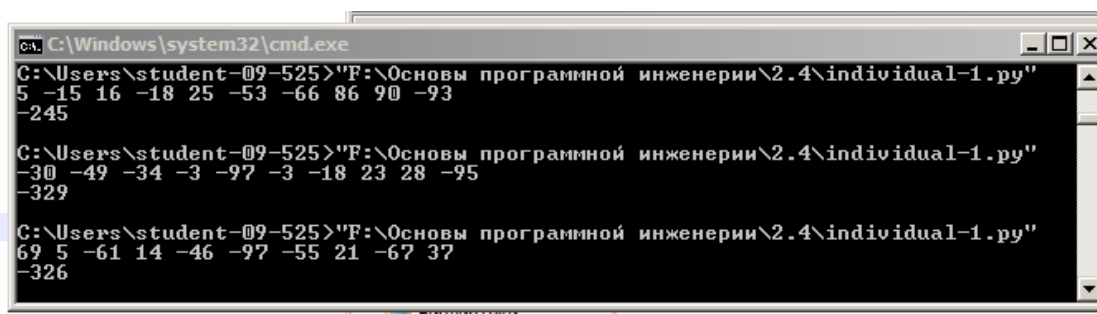


```
C:\Windows\system32\cmd.exe
C:\Users\student-09-525>"F:\Основы программной инженерии\2.4\example-2.py"
5 15 16 18 25 53 66 86 90 93
8

C:\Users\student-09-525>"F:\Основы программной инженерии\2.4\example-2.py"
-30 -49 -34 -3 -97 -3 -18 23
0

C:\Users\student-09-525>"F:\Основы программной инженерии\2.4\example-2.py"
28 95 69 5 61 14 46 97 55 21 67 37
3
```

7. Создаём файл «individual-1.py», в котором пользователю нужно ввести список из 10 элементов, и программа должна найти сумму отрицательных элементов и вывести её на экран.

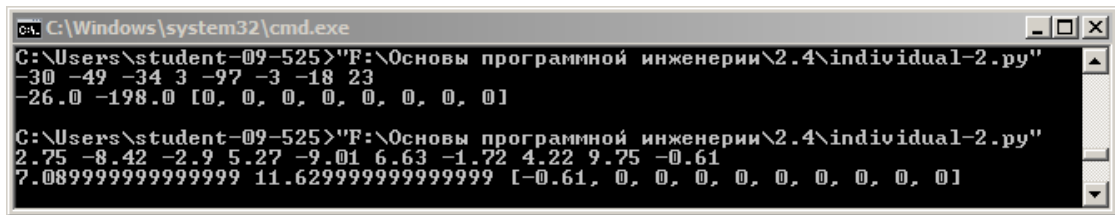


```
C:\Windows\system32\cmd.exe
C:\Users\student-09-525>"F:\Основы программной инженерии\2.4\individual-1.py"
5 -15 16 -18 25 -53 -66 86 90 -93
-245

C:\Users\student-09-525>"F:\Основы программной инженерии\2.4\individual-1.py"
-30 -49 -34 -3 -97 -3 -18 23 28 -95
-329

C:\Users\student-09-525>"F:\Основы программной инженерии\2.4\individual-1.py"
69 5 -61 14 -46 -97 -55 21 -67 37
-326
```

8. Создаём файл «individual-2.py», в котором пользователю нужно ввести список, состоящий из вещественных чисел, и программа должна найти сумму нечётных элементов и сумму элементов, находящихся между первым и последним отрицательным элементами, убрать все элементы, по модулю большие 1, и поместить в конец списка вместо недостающих элементов нули.



```
C:\Windows\system32\cmd.exe
C:\Users\student-09-525>"F:\Основы программной инженерии\2.4\individual-2.py"
-30 -49 -34 3 -97 -3 -18 23
-26.0 -198.0 [0, 0, 0, 0, 0, 0, 0, 0]

C:\Users\student-09-525>"F:\Основы программной инженерии\2.4\individual-2.py"
2.75 -8.42 -2.9 5.27 -9.01 6.63 -1.72 4.22 9.75 -0.61
7.089999999999999 11.629999999999999 [-0.61, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

9. Сохраняем лабораторную работу в качестве PDF-файла и выполняем коммит файлов в репозиторий Git в ветку разработки, сливаем её с веткой main и отправляем изменения на сервер GitHub.

```
@RomanGorchakov →/workspaces/Py4 (develop) $ git add .
● @RomanGorchakov →/workspaces/Py4 (develop) $ git commit -m "Python files"
[develop 8153382] Python files
4 files changed, 106 insertions(+)
create mode 100644 example-1.py
create mode 100644 example-2.py
create mode 100644 individual-1.py
create mode 100644 individual-2.py
● @RomanGorchakov →/workspaces/Py4 (develop) $ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
● @RomanGorchakov →/workspaces/Py4 (main) $ git merge develop
Updating fd4ba9d..8153382
Fast-forward
 example-1.py | 18 +++++
 example-2.py | 29 +++++
 individual-1.py | 18 +++++
 individual-2.py | 41 +++++
4 files changed, 106 insertions(+)
create mode 100644 example-1.py
create mode 100644 example-2.py
create mode 100644 individual-1.py
create mode 100644 individual-2.py
● @RomanGorchakov →/workspaces/Py4 (main) $ git push -u
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 2 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 1.47 KiB | 1.47 MiB/s, done.
Total 6 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/RomanGorchakov/Py4
fd4ba9d..8153382 main -> main
branch 'main' set up to track 'origin/main'.
○ @RomanGorchakov →/workspaces/Py4 (main) $
```

main

1 Branch

0 Tags

Go to file

Add file

Code

RomanGorchakov Python files

8153382 · 1 minute ago

4 Commits

.gitignore	Create .gitignore	last week
LICENSE	Create LICENSE	last week
README.md	Create README.md	last week
example-1.py	Python files	1 minute ago
example-2.py	Python files	1 minute ago
individual-1.py	Python files	1 minute ago
individual-2.py	Python files	1 minute ago

README

MIT license

Информация обо мне

Имя: Роман.

Фамилия: Горчаков.

Отчество: Владимирович.

Название группы: ПИЖ-6-о-22-1.

Контрольные вопросы

1. Что такое списки в языке Python?

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

2. Как осуществляется создание списка в Python?

Для создания списка нужно заключить элементы в квадратные скобки.

3. Как организовано хранение списков в оперативной памяти?

Список является изменяемым типом данных. При его создании в памяти резервируется область, которую можно условно назвать некоторым «контейнером», в котором хранятся ссылки на другие элементы данных в памяти. В отличие от таких типов данных как число или строка, содержимое «контейнера» списка можно менять.

4. Каким образом можно перебрать все элементы списка?

Читать элементы списка можно с помощью цикла for.

5. Какие существуют арифметические операции со списками?

Объединение списков (+), повторение списка n раз (*).

6. Как проверить есть ли элемент в списке?

Для того, чтобы проверить, есть ли заданный элемент в списке Python необходимо использовать оператор `in`.

7. Как определить число вхождений заданного элемента в списке?

Метод `count` можно использовать для определения числа сколько раз данный элемент встречается в списке.

8. Как осуществляется добавление (вставка) элемента в список?

Метод `insert` можно использовать, чтобы вставить элемент в список.

9. Как выполнить сортировку списка?

Для сортировки списка нужно использовать метод `sort`.

10. Как удалить один или несколько элементов из списка?

Удалить элемент можно, написав его индекс в методе `pop`. Можно удалить несколько элементов с помощью оператора среза.

11. Что такое списковое включение и как с его помощью осуществлять обработку списков?

List Comprehensions чаще всего на русский язык переводят как абстракция списков или списковое включение, является частью синтаксиса языка, которая предоставляет простой способ построения списков.

В языке Python есть две очень мощные функции для работы с коллекциями: `map` и `filter`. Они позволяют использовать функциональный стиль программирования, не прибегая к помощи циклов, для работы с такими типами как `list`, `tuple`, `set`, `dict` и т.п. Списковое включение позволяет обойтись без этих функций.

12. Как осуществляется доступ к элементам списков с помощью срезов?

Срез задается тройкой чисел, разделенных запятой: `start:stop:step`. `Start` – позиция с которой нужно начать выборку, `stop` – конечная позиция, `step` – шаг. При этом необходимо помнить, что выборка не включает элемент, определяемый `stop`.

13. Какие существуют функции агрегации для работы со списками?

Для работы со списками Python предоставляет следующие функции:

- `len(L)` – получить число элементов в списке `L`.
- `min(L)` – получить минимальный элемент списка `L`.
- `max(L)` – получить максимальный элемент списка `L`.
- `sum(L)` – получить сумму элементов списка `L`, если список `L` содержит

только числовые значения.

14. Как создать копию списка?

Для создания копии списка необходимо использовать либо метод `copy`, либо использовать оператор среза.

15. Самостоятельно изучите функцию `sorted` языка Python. В чем ее отличие от метода `sort` списков?

Функция `sorted` возвращает новый отсортированный список, который получен из итерируемого объекта, который был передан как аргумент. Функция также поддерживает дополнительные параметры, которые позволяют управлять сортировкой.

Метод `sort` применяется к спискам (в отличие от функции `sorted()`, которая применяется к любым итерируемым объектам).