

## Лабораторная работа 2.7

Тема: Работа с множествами в языке Python.

Цель работы: приобретение навыков по работе с множествами 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.](#)

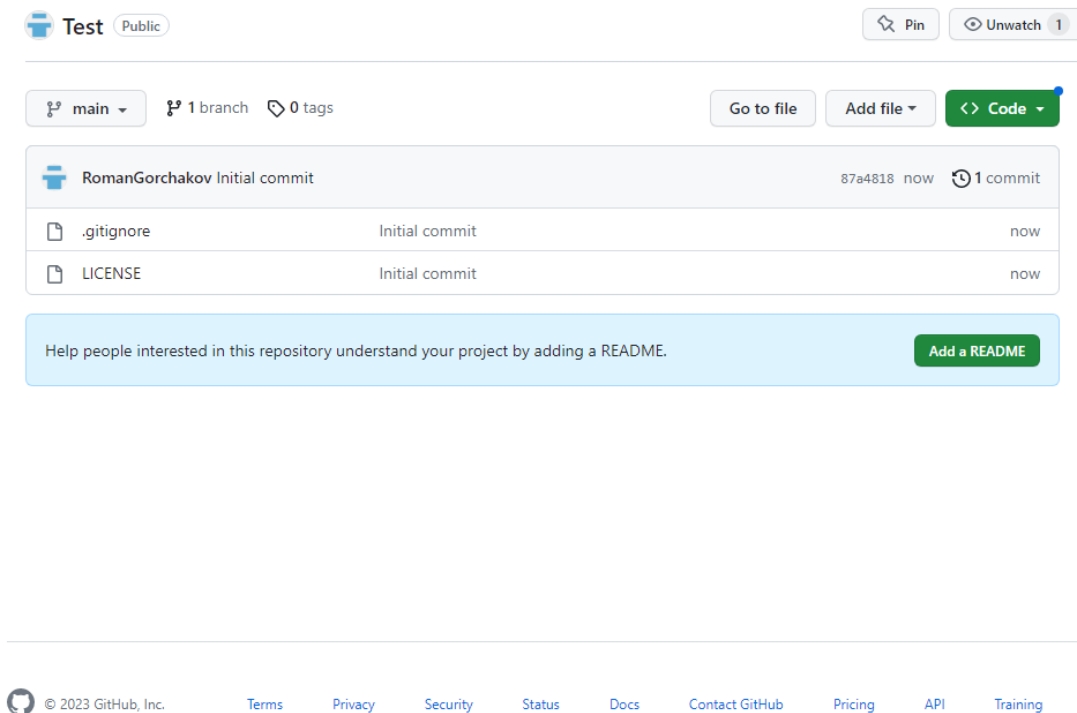
Choose a license

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
	<a href="#">Python.gitignore</a>	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/Py7 (main) $ git checkout -b develop
Switched to a new branch 'develop'
● @RomanGorchakov → /workspaces/Py7 (develop) $ git branch feature_branch
● @RomanGorchakov → /workspaces/Py7 (develop) $ git branch release/1.0.0
● @RomanGorchakov → /workspaces/Py7 (develop) $ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
● @RomanGorchakov → /workspaces/Py7 (main) $ git branch hotfix
● @RomanGorchakov → /workspaces/Py7 (main) $ git checkout develop
Switched to branch 'develop'
○ @RomanGorchakov → /workspaces/Py7 (develop) $
```

5. Создаём файл «example.py», в котором программа должна определить результат выполнения операций над множествами.

```
x = {'e', 'o', 'k', 'j', 'd'}  
y = {'y', 'h', 'g', 'c', 'o', 'v', 'f'}  
  
...Program finished with exit code 0  
Press ENTER to exit console.□
```

6. Создаём файл «vowel.py», в котором пользователю нужно ввести строку, а программа подсчитывает количество гласных в ней и выводит его на экран.

```
Введите строку: jtrjrawqfl  
1  
  
...Program finished with exit code 0  
Press ENTER to exit console.□
```

7. Создаём файл «same.py», в котором пользователю нужно ввести две строки, а программа определяет общие символы в них.

```
Введите первую строку: kzrfmqxi  
Введите вторую строку: uzrnmcgr  
z 1  
r 2  
m 1  
  
...Program finished with exit code 0  
Press ENTER to exit console.□
```

8. Создаём файл «individual.py», в котором программа должна определить результат выполнения операций над множествами.

```
x = {'j'}  
y = {'y', 'd', 'f', 'x', 'q', 'e', 'g', 'p', 'o', 'u', 's', 'm'}  
  
...Program finished with exit code 0  
Press ENTER to exit console.□
```

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

```

@RomanGorchakov →/workspaces/Py7 (develop) $ git add .
● @RomanGorchakov →/workspaces/Py7 (develop) $ git commit -m "Python files"
[develop 6a2cba7] Python files
4 files changed, 77 insertions(+)
create mode 100644 example.py
create mode 100644 individual.py
create mode 100644 same.py
create mode 100644 vowel.py
● @RomanGorchakov →/workspaces/Py7 (develop) $ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
● @RomanGorchakov →/workspaces/Py7 (main) $ git merge develop
Updating 0fb9b83..6a2cba7
Fast-forward
 example.py      | 19 +++++++++++++++++++++
 individual.py   | 19 +++++++++++++++++++++
 same.py         | 21 ++++++++++++++++++++++
 vowel.py        | 18 ++++++++++++++++++++
4 files changed, 77 insertions(+)
create mode 100644 example.py
create mode 100644 individual.py
create mode 100644 same.py
create mode 100644 vowel.py
● @RomanGorchakov →/workspaces/Py7 (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 0), reused 0 (delta 0), pack-reused 0
To https://github.com/RomanGorchakov/Py7
 0fb9b83..6a2cba7  main -> main
branch 'main' set up to track 'origin/main'.
○ @RomanGorchakov →/workspaces/Py7 (main) $

```

main
1 Branch
0 Tags
Go to file
Add file
Code

RomanGorchakov Python files
6a2cba7 · 6 minutes ago
4 Commits

.gitignore	Create .gitignore	2 weeks ago
LICENSE	Create LICENSE	2 weeks ago
README.md	Create README.md	2 weeks ago
example.py	Python files	6 minutes ago
individual.py	Python files	6 minutes ago
same.py	Python files	6 minutes ago
vowel.py	Python files	6 minutes 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 множества в языке Python?

Множеством в языке программирования Python называется неупорядоченная совокупность уникальных значений. В качестве элементов этого набора данных могут выступать любые неизменяемые объекты, такие как числа, символы, строки. В отличие от массивов и списков, порядок следования значений не учитывается при обработке его содержимого.

### 2. Как осуществляется создание множеств в Python?

Перед тем как начать работу с множеством, необходимо для начала его создать. Сделать это можно, просто присвоив переменной последовательность значений, выделив их фигурными скобками.

Существует и другой способ создания множеств, который подразумевает использование вызова `set`. Аргументом этой функции может быть набор неких данных или даже строка с текстом.

### 3. Как проверить присутствие/отсутствие элемента в множестве?

Для проверки, есть ли данное значение в множестве, используется `in`, а для проверки отсутствия элемента – `not in`.

### 4. Как выполнить перебор элементов множества?

```
For a in mn:  
    print(a)
```

### 5. Что такое set comprehension?

Для создания множества можно в Python воспользоваться генератором, позволяющих заполнять списки, а также другие наборы данных с учетом неких условий.

### 6. Как выполнить добавление элемента во множество?

Чтобы внести новые значения, потребуется вызывать метод `add`. Аргументом в данном случае будет добавляемый элемент последовательности.

### 7. Как выполнить удаление одного или всех элементов множества?

Для удаления элементов из множества используются следующие функции в Python (кроме очистки, которая будет рассмотрена ниже):

- `remove` – удаление элемента с генерацией исключения в случае, если такого элемента нет;
- `discard` – удаление элемента без генерации исключения, если элемент отсутствует;
- `pop` – удаление первого элемента, генерируется исключение при попытке удаления из пустого множества.

8. Как выполняются основные операции над множествами: объединение, пересечение, разность?

Чтобы объединить все элементы двух разных множеств, стоит воспользоваться методом `union` на одном из объектов. Чтобы найти общие элементы для двух разных множеств, следует применить функцию `intersection`, принимающую в качестве аргумента один из наборов данных. Чтобы вычислить разность для двух разных множеств, необходимо воспользоваться методом `difference`.

9. Как определить, что некоторое множество является надмножеством или подмножеством другого множества?

Чтобы выяснить, является ли множество `a` подмножеством `b`, стоит попробовать вывести на экран результат выполнения метода `issubset`. Чтобы узнать, является ли множество `a` надмножеством `b`, необходимо вызвать метод `issuperset` и вывести результат его работы на экран.

10. Каково назначение множеств `frozenset`?

Множество, содержимое которого не поддается изменению имеет тип `frozenset`. Значения из этого набора нельзя удалить, как и добавить новые. Поскольку содержимое `frozenset` должно всегда оставаться статичным, перечень функций, с которыми такое множество может взаимодействовать, имеет ограничения.

11. Как осуществляется преобразование множеств в строку, список, словарь?

Для преобразования множества в строку используется конкатенация текстовых значений, которую обеспечивает функция `join`. В этом случае ее



аргументом является набор данных в виде нескольких строк. Запятая в кавычках выступает в качестве символа, разделяющего значения.

Чтобы получить из множества словарь, следует передать функции `dict` набор из нескольких пар значений, в каждом из которых будет находиться ключ.

По аналогии с предыдущими преобразованиями можно получить список неких объектов. На этот раз используется вызов `list`, получающий в качестве аргумента множество `a`.