

Лабораторная работа 2.9

Тема: Рекурсия в языке Python.

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

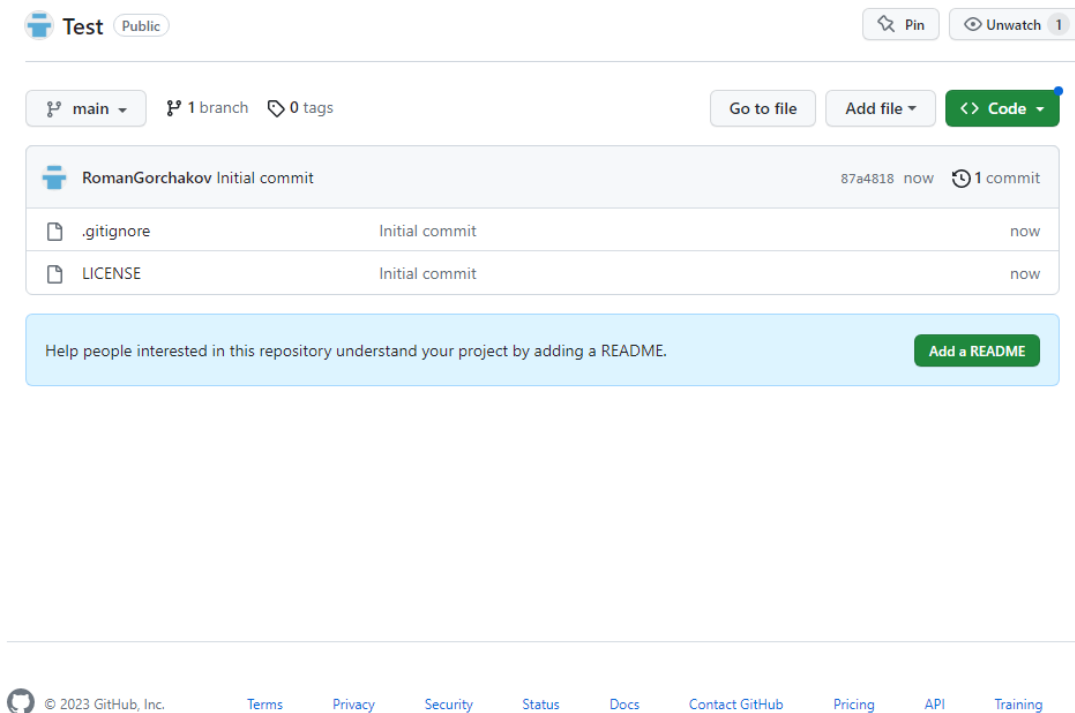
Choose a license

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/Py9 (main) $ git checkout -b develop
  Switched to a new branch 'develop'
● @RomanGorchakov →/workspaces/Py9 (develop) $ git branch feature_branch
● @RomanGorchakov →/workspaces/Py9 (develop) $ git branch release/1.0.0
● @RomanGorchakov →/workspaces/Py9 (develop) $ git checkout main
  Switched to branch 'main'
  Your branch is up to date with 'origin/main'.
● @RomanGorchakov →/workspaces/Py9 (main) $ git branch hotfix
● @RomanGorchakov →/workspaces/Py9 (main) $ git checkout develop
  Switched to branch 'develop'
○ @RomanGorchakov →/workspaces/Py9 (develop) $
```

5. Создаём файл «example.py», в котором программа выполняет хвостовую рекурсию.

6. Создаём файл «individual.py», в котором пользователю нужно ввести число N, а программа высчитывает количество перестановок чисел от 1 до N.

```
Введите число: 3
[[1, 2, 3], [2, 1, 3], [2, 3, 1], [1, 3, 2], [3, 1, 2], [3, 2, 1]]

...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/Py9 (develop) $ git add .
● @RomanGorchakov →/workspaces/Py9 (develop) $ git commit -m "Python files"
[develop 86f7c63] Python files
 2 files changed, 69 insertions(+)
 create mode 100644 example.py
 create mode 100644 individual.py
● @RomanGorchakov →/workspaces/Py9 (develop) $ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
● @RomanGorchakov →/workspaces/Py9 (main) $ git merge develop
Updating db9481f..86f7c63
Fast-forward
 example.py | 43 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 individual.py | 26 +++++++++++++++++++++++++++++++++++++
 2 files changed, 69 insertions(+)
 create mode 100644 example.py
 create mode 100644 individual.py
● @RomanGorchakov →/workspaces/Py9 (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), 1.36 KiB | 1.36 MiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/RomanGorchakov/Py9
 db9481f..86f7c63 main -> main
 branch 'main' set up to track 'origin/main'.
○ @RomanGorchakov →/workspaces/Py9 (main) $ □
```

main

1 Branch

0 Tags

Go to file

Add file

Code

RomanGorchakov Python files

86f7c63 · 1 minute ago

4 Commits

.gitignore	Create .gitignore	21 minutes ago
LICENSE	Create LICENSE	2 weeks ago
README.md	Create README.md	2 weeks ago
example.py	Python files	1 minute ago
individual.py	Python files	1 minute ago

README

MIT license

Информация обо мне

Имя: Роман.

Фамилия: Горчаков.

Отчество: Владимирович.

Название группы: ПИЖ-б-о-22-1.

Контрольные вопросы

1. Для чего нужна рекурсия?

Функция может содержать вызов других функций. В том числе процедура может вызвать саму себя. Никакого парадокса здесь нет – компьютер лишь последовательно выполняет встретившиеся ему в программе команды и, если встречается вызов процедуры, просто начинает выполнять эту функцию. Без разницы, какая функция дала команду это делать.

2. Что называется базой рекурсии?

Для того чтобы рекурсия не продолжалась "бесконечно", необходимо условие выхода из рекурсии (база рекурсии).

3. Самостоятельно изучите что является стеком программы. Как используется стек программы при вызове функций?

Стек – вид структуры данных, в котором элементы упорядочены и добавление или удаление элементов происходит с верхней части стека.

При вызове подпрограммы в стек заносится адрес возврата – адрес в памяти следующей инструкции приостанавливаемой программы, а управление передается подпрограмме. При последующем вложенном или рекурсивном вызове в стек заносится очередной адрес возврата и так далее.

4. Как получить текущее значение максимальной глубины рекурсии в языке Python?

Чтобы проверить текущие параметры лимита, нужно запустить `sys.getrecursionlimit()`.

5. Что произойдет, если число рекурсивных вызовов превысит максимальную глубину рекурсии в языке Python?

Когда предел достигнут, возникает исключение `RuntimeError: Maximum Recursion Depth Exceeded`.

6. Как изменить максимальную глубину рекурсии в языке Python?

Можно изменить предел глубины рекурсии с помощью вызова `sys.setrecursionlimit(limit)`.

7. Каково назначение декоратора `lru_cache`?

Полезным инструментом является декоратор `lru_cache`, который можно использовать для уменьшения количества лишних вычислений.

8. Что такое хвостовая рекурсия? Как проводится оптимизация хвостовых вызовов?

Хвостовая рекурсия – это частный случай рекурсии, при котором любой рекурсивный вызов является последней операцией перед возвратом из функции. Подобный вид рекурсии примечателен тем, что может быть легко заменён на итерацию путём формальной и гарантированно корректной перестройки кода функции.

Оптимизация хвостовой рекурсии путём преобразования её в плоскую итерацию реализована во многих оптимизирующих компиляторах. В некоторых функциональных языках программирования спецификация гарантирует обязательную оптимизацию хвостовой рекурсии.

Типовой механизм реализации вызова функции основан на сохранении адреса возврата, параметров и локальных переменных функции в стеке и выглядит следующим образом:

1. В точке вызова в стек помещаются параметры, передаваемые функции, и адрес возврата.
2. Вызываемая функция в ходе работы размещает в стеке собственные локальные переменные.
3. По завершении вычислений функция очищает стек от своих локальных переменных, записывает результат (обычно — в один из регистров процессора).
4. Команда возврата из функции считывает из стека адрес возврата и выполняет переход по этому адресу. Либо непосредственно перед, либо сразу после возврата из функции стек очищается от параметров.