

Лабораторная работа 2.17

Тема: Разработка приложений с интерфейсом командной строки (CLI) в Python3.

Цель работы: приобретение построения приложений с интерфейсом командной строки 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.](#)

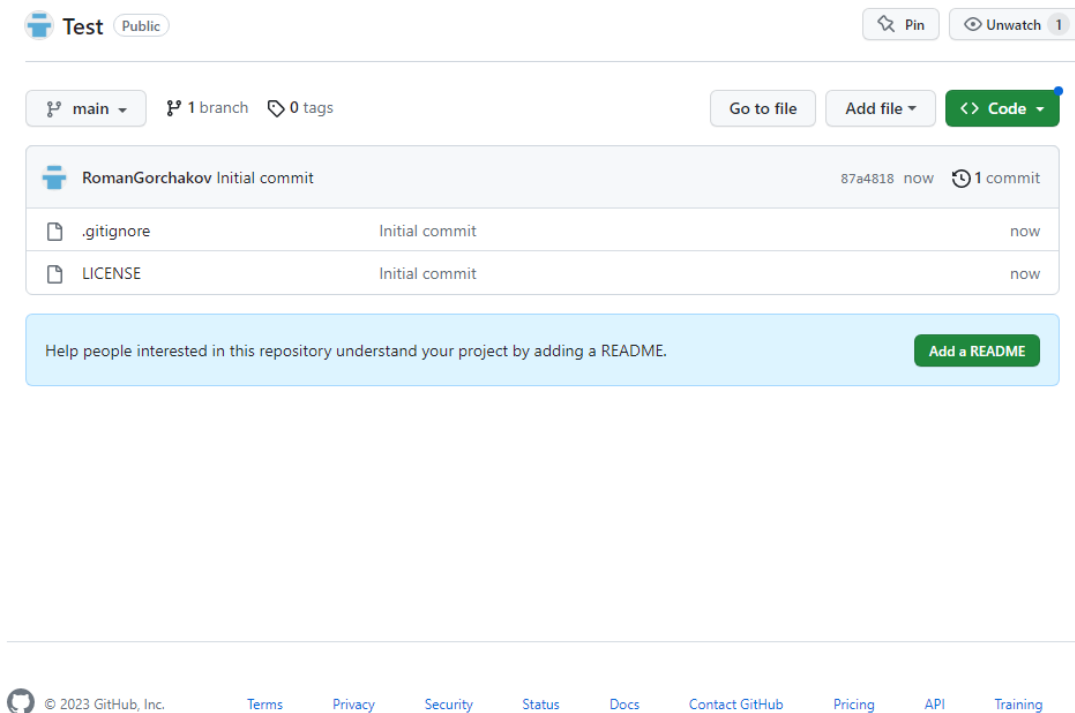
Choose a license

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. В окне «Codespace» выбираем опцию «Create codespace on main». Откроется терминал, куда мы введём команду «git clone», чтобы клонировать свой репозиторий. После этого организуем репозиторий в соответствии с моделью ветвления Git-flow. Для этого введём в терминал команды: «git checkout –b 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. Создаём файл .pre-commit-config.yaml и environment.yml.

```
● @RomanGorchakov → /workspaces/Py17 (main) $ git clone https://github.com/RomanGorchakov/Py17.git ClonedPy3.git
Cloning into 'ClonedPy3.git'...
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 9 (delta 1), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (9/9), 4.80 KiB | 4.80 MiB/s, done.
Resolving deltas: 100% (1/1), done.
● @RomanGorchakov → /workspaces/Py17 (main) $ git checkout -b develop
Switched to a new branch 'develop'
● @RomanGorchakov → /workspaces/Py17 (develop) $ git branch feature_branch
● @RomanGorchakov → /workspaces/Py17 (develop) $ git branch release/1.0.0
● @RomanGorchakov → /workspaces/Py17 (develop) $ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
● @RomanGorchakov → /workspaces/Py17 (main) $ git branch hotfix
● @RomanGorchakov → /workspaces/Py17 (main) $ git checkout develop
Switched to branch 'develop'
● @RomanGorchakov → /workspaces/Py17 (develop) $
```

```

97.9/97.9 kB 2.6 MB/s eta 0:00:00
Downloading mypy_extensions-1.0.0-py3-none-any.whl (4.7 kB)
Downloading pathspec-0.12.1-py3-none-any.whl (31 kB)
Installing collected packages: pathspec, mypy_extensions, click, black
Successfully installed black-24.4.2 click-8.1.7 mypy_extensions-1.0.0 pathspec-0.12.1
• @RomanGorchakov → /workspaces/Py17 (develop) $ pip install flake8
Collecting flake8
  Downloading flake8-7.0.0-py2.py3-none-any.whl.metadata (3.8 kB)
Collecting mccabe<0.8.0,>=0.7.0 (from flake8)
  Downloading mccabe-0.7.0-py2.py3-none-any.whl.metadata (5.0 kB)
Collecting pycodestyle<2.12.0,>=2.11.0 (from flake8)
  Downloading pycodestyle-2.11.1-py2.py3-none-any.whl.metadata (4.5 kB)
Collecting pyflakes<3.3.0,>=3.2.0 (from flake8)
  Downloading pyflakes-3.2.0-py2.py3-none-any.whl.metadata (3.5 kB)
  Downloading flake8-7.0.0-py2.py3-none-any.whl (57 kB)
57.6/57.6 kB 1.5 MB/s eta 0:00:00
  Downloading mccabe-0.7.0-py2.py3-none-any.whl (7.3 kB)
  Downloading pycodestyle-2.11.1-py2.py3-none-any.whl (31 kB)
  Downloading pyflakes-3.2.0-py2.py3-none-any.whl (62 kB)
62.7/62.7 kB 1.6 MB/s eta 0:00:00
Installing collected packages: pyflakes, pycodestyle, mccabe, flake8
Successfully installed flake8-7.0.0 mccabe-0.7.0 pycodestyle-2.11.1 pyflakes-3.2.0
• @RomanGorchakov → /workspaces/Py17 (develop) $ isort pre-commit step > .pre-commit-config.yaml
• @RomanGorchakov → /workspaces/Py17 (develop) $ pre-commit sample-config > .pre-commit-config.yaml
• @RomanGorchakov → /workspaces/Py17 (develop) $ isort pre-commit step
Broken 2 paths
• @RomanGorchakov → /workspaces/Py17 (develop) $ conda env export > environment.yml
• @RomanGorchakov → /workspaces/Py17 (develop) $ 

```

5. Создаём файл «example.py», в котором нужно реализовать интерфейс командной строки CLI для примера из лабораторной работы 2.16.

```

Командная строка

H:\Основы программной инженерии\2.17>python example.py add data.json --name="Иванов Иван" --post="Директор" --year=2007

H:\Основы программной инженерии\2.17>python example.py add data.json --name="Петров Пётр" --post="Бухгалтер" --year=2010

H:\Основы программной инженерии\2.17>python example.py add data.json --name="Сидоров Сидор" --post="Главный Инженер" --year=2012

H:\Основы программной инженерии\2.17>python example.py display data.json
+-----+-----+-----+-----+
| № | Ф.И.О. | Должность | Год |
+-----+-----+-----+-----+
| 1 | Иванов Иван | Директор | 2007 |
| 2 | Петров Пётр | Бухгалтер | 2010 |
| 3 | Сидоров Сидор | Главный Инженер | 2012 |
+-----+-----+-----+-----+

H:\Основы программной инженерии\2.17>python example.py select data.json --period=12
+-----+-----+-----+-----+
| № | Ф.И.О. | Должность | Год |
+-----+-----+-----+-----+
| 1 | Иванов Иван | Директор | 2007 |
+-----+-----+-----+-----+

H:\Основы программной инженерии\2.17>

```

```
[
  {
    "name": "Иванов Иван",
    "post": "Директор",
    "year": 2007
  },
  {
    "name": "Петров Пётр",
    "post": "Бухгалтер",
    "year": 2010
  },
  {
    "name": "Сидоров Сидор",
    "post": "Главный Инженер",
    "year": 2012
  }
]
```

6. Создаём файл «individual.py», в котором нужно реализовать интерфейс командной строки CLI. Написать программу, выполняющую следующие действия: ввод с клавиатуры данных в список, состоящий из словарей заданной структуры; записи должны быть упорядочены по возрастанию номера рейса; вывод на экран номеров рейсов и типов самолетов, вылетающих в пункт назначения, название которого совпало с названием, введенным с клавиатуры; если таких рейсов нет, выдать на дисплей соответствующее сообщение. Необходимо также проследить за тем, чтобы файлы генерируемый этой программой не попадали в репозиторий лабораторной работы.

```
Командная строка

H:\Основы программной инженерии\2.17>python individual.py add schedule.json --ra
ce="Калининград" --number=4449 --type=90

H:\Основы программной инженерии\2.17>python individual.py add schedule.json --ra
ce="Казань" --number=3780 --type=1

H:\Основы программной инженерии\2.17>python individual.py add schedule.json --ra
ce="Москва" --number=8850 --type=9

H:\Основы программной инженерии\2.17>python individual.py add schedule.json --ra
ce="Новоалександровск" --number=3234 --type=62

H:\Основы программной инженерии\2.17>python individual.py add schedule.json --ra
ce="Санкт-Петербург" --number=9864 --type=48

H:\Основы программной инженерии\2.17>python individual.py add schedule.json --ra
ce="Ставрополь" --number=9120 --type=82

H:\Основы программной инженерии\2.17>python individual.py display schedule.json
+-----+-----+-----+-----+
| No | Пункт назначения | Номер рейса | Тип самолёта |
+-----+-----+-----+-----+
| 1 | Калининград | 4449 | 90 |
| 2 | Казань | 3780 | 1 |
| 3 | Москва | 8850 | 9 |
| 4 | Новоалександровск | 3234 | 62 |
| 5 | Санкт-Петербург | 9864 | 48 |
| 6 | Ставрополь | 9120 | 82 |
+-----+-----+-----+-----+

H:\Основы программной инженерии\2.17>
```

```
[
  {
    "race": "Калининград",
    "number": 4449,
    "type": 90
  },
  {
    "race": "Казань",
    "number": 3780,
    "type": 1
  },
  {
    "race": "Москва",
    "number": 8850,
    "type": 9
  },
  {
    "race": "Новоалександровск",
    "number": 3234,
    "type": 62
  },
  {
    "race": "Санкт-Петербург",
    "number": 9864,
    "type": 48
  },
  {
    "race": "Ставрополь",
    "number": 9120,
    "type": 82
  }
]
```

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

```
create mode 100644 "code/\320\230\320\275\320\264\320\270\320\262\320\270\320\264\321\203\320\260\320\273\321\214\320\275\320\276\320\265 \320\267\320\260\320\264\320\260\320\275\320\270\320\265\packet\get_plane.py"
create mode 100644 "code/\320\230\320\275\320\264\320\270\320\262\320\270\320\264\321\203\320\260\320\273\321\214\320\275\320\276\320\265 \320\267\320\260\320\264\320\260\320\275\320\270\320\265\packet\load_plane.py"
create mode 100644 "code/\320\230\320\275\320\264\320\270\320\262\320\270\320\264\321\203\320\260\320\273\321\214\320\275\320\276\320\265 \320\267\320\260\320\264\320\260\320\275\320\270\320\265\packet\packet.py"
create mode 100644 "code/\320\230\320\275\320\264\320\270\320\262\320\270\320\264\321\203\320\260\320\273\321\214\320\275\320\276\320\265 \320\267\320\260\320\264\320\260\320\275\320\270\320\265\packet\save_plane.py"
create mode 100644 "code/\320\230\320\275\320\264\320\270\320\262\320\270\320\264\321\203\320\260\320\273\321\214\320\275\320\276\320\265 \320\267\320\260\320\264\320\260\320\275\320\270\320\265\packet\show_plane.py"
create mode 100644 "code/\320\230\320\275\320\264\320\270\320\262\320\270\320\264\321\203\320\260\320\273\321\214\320\275\320\276\320\265 \320\267\320\260\320\264\320\260\320\275\320\270\320\265\schedule.json"
create mode 100644 "code/\320\237\321\200\320\270\320\274\320\265\321\200\data.json"
create mode 100644 "code/\320\237\321\200\320\270\320\274\320\265\321\200/example.py"
create mode 100644 "doc/\320\233\320\2402.17_\320\223\320\276\321\200\321\207\320\260\320\272\320\276\320\262\320\240\320\222.pdf"
create mode 100644 environment.yml
@RomanGorchakov →/workspaces/Py17 (main) $ git push -u
Enumerating objects: 22, done.
Counting objects: 100% (22/22), done.
Delta compression using up to 2 threads
Compressing objects: 100% (20/20), done.
Writing objects: 100% (21/21), 480.55 KiB | 20.02 MiB/s, done.
Total 21 (delta 3), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (3/3), done.
To https://github.com/RomanGorchakov/Py17
 e69f899..a8a0ecc main -> main
branch 'main' set up to track 'origin/main'.
@RomanGorchakov →/workspaces/Py17 (main) $
```

RomanGorchakov CLIa8a0ecc · 1 minute ago🕒 4 Commits		
code	CLI	1 minute ago
doc	CLI	1 minute ago
.gitignore	Create .gitignore	16 minutes ago
.pre-commit-config.yaml	CLI	1 minute ago
LICENSE	Create LICENSE	15 minutes ago
README.md	Create README.md	16 minutes ago
environment.yml	CLI	1 minute ago

README🔗 MIT license✎

Лабораторная работа 2.17. Тема: Разработка приложений с интерфейсом командной строки (CLI) в Python3.

Контрольные вопросы

1. В чем отличие терминала и консоли?

Терминал – это устройство или ПО, выступающее посредником между человеком и вычислительной системой. Обычно данный термин используется, когда точка доступа к системе вынесена в отдельное физическое устройство и предоставляет свой пользовательский интерфейс на основе внутреннего интерфейса (например, сетевых протоколов).

Консоль – это исторически реализация терминала с клавиатурой и текстовым дисплеем. В настоящее время это слово часто используется как синоним сеанса работы или окна оболочки командной строки.

2. Что такое консольное приложение?

Консольное приложение – это вид ПО, разработанный с расчётом на работу внутри оболочки командной строки, т.е. опирающийся на текстовый ввод-вывод.

3. Какие существуют средства языка программирования Python для построения приложений командной строки?

Операционная среда – это интерфейс, предоставляемый пользователю или программе операционной системой. В частности, пользовательский интерфейс является частью операционной среды.

Командная строка – это принцип организации пользовательского интерфейса на основе ввода текстовых команд с клавиатуры и текстового вывода результатов на экран

Оболочка командной строки – это программное обеспечение, отвечающее за поддержку командной строки (обычно это компонент ОС, но может быть и сторонним ПО). Оболочка командной строки предоставляет собственное окружение: «переменные среды» (глобальные и локальные для текущего сеанса) и интерпретатор текстовых команд.

Пакетный файл или сценарий – это содержащий команды оболочки файл, который можно запустить на исполнение как исполняемый файл.

4. Какие особенности построение CLI с использованием модуля sys?

Модуль `sys` – базовый модуль, который с самого начала поставлялся с Python. Он использует подход, очень похожий на библиотеку C, с использованием `argc` и `argv` для доступа к аргументам. Модуль `sys` реализует аргументы командной строки в простой структуре списка с именем `sys.argv`.

Каждый элемент списка представляет собой единственный аргумент. Первый элемент в списке `sys.argv [0]` – это имя скрипта Python. Остальные элементы списка, от `sys.argv [1]` до `sys.argv [n]`, являются аргументами командной строки с 2 по n. В качестве разделителя между аргументами используется пробел. Значения аргументов, содержащие пробел, должны быть заключены в кавычки, чтобы их правильно проанализировал `sys`. Эквивалент `argc` – это просто количество элементов в списке. Чтобы получить это значение, используйте оператор `len()`.

5. Какие особенности построение CLI с использованием модуля `getopt`?

Основанный на функции C `getopt`, он позволяет использовать как короткие, так и длинные варианты, включая присвоение значений. На практике для правильной обработки входных данных требуется модуль `sys`. Для этого необходимо заранее загрузить как модуль `sys`, так и модуль `getopt`. Затем из списка входных параметров мы удаляем первый элемент списка и сохраняем оставшийся список аргументов командной строки в переменной с именем `arguments_list`.

6. Какие особенности построение CLI с использованием модуля `argparse` ?

Начиная с версий Python 2.7 и Python 3.2, в набор стандартных библиотек была включена библиотека `argparse` для обработки аргументов (параметров, ключей) командной строки.

Для начала рассмотрим, что интересного предлагает `argparse`:

- анализ аргументов `sys.argv`;
- конвертирование строковых аргументов в объекты Вашей программы и работа с ними;
- форматирование и вывод информативных подсказок.

Как заявляют разработчики `argparse`, библиотеки `getopt` и `optparse` уступают `argparse` по нескольким причинам:

- обладая всей полнотой действий с обычными параметрами командной строки, они не умеют обрабатывать позиционные аргументы (positional arguments). Позиционные аргументы – аргументы, влияющие на работу программы, в зависимости от порядка, в котором они в эту программу передаются. Простейший пример — программа `cp`, имеющая минимум 2 таких аргумента («`cp source destination`»);

- `argparse` дает на выходе более качественные сообщения о подсказке при минимуме затрат (в этом плане при работе с `optparse` часто можно наблюдать некоторую избыточность кода);

- `argparse` дает возможность программисту устанавливать для себя, какие символы являются параметрами, а какие нет. В отличие от него, `optparse` считает опции с синтаксисом наподобие `"-pf`, `-file`, `+rgb`, `/f` и т.п. «внутренне противоречивыми» и «не поддерживается `optpars` 'ом и никогда не будет»;

- `argparse` даст Вам возможность использовать несколько значений переменных у одного аргумента командной строки (nargs);

- `argparse` поддерживает субкоманды (subcommands). Это когда основной парсер отправляет к другому (субпарсеру), в зависимости от аргументов на входе.