

Лабораторная работа 4.2

Тема: Перегрузка операторов в языке Python.

Цель работы: приобретение навыков по перегрузке операторов при написании программ с помощью языка программирования Python версии 3.x.

Ссылка на GitHub: https://github.com/RomanGorchakov/Py3_3


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 *	Repository name *
 RomanGorchakov ▾	/ 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

▾

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)


Choose a 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

 **Test** Public

 Pin  Unwatch 1

main 1 branch 0 tags

Go to file Add file <> Code

 **RomanGorchakov** Initial commit 87a4818 now 1 commit

 .gitignore Initial commit now


















 LICENSE Initial commit now

Help people interested in this repository understand your project by adding a README.

Add a README

 © 2023 GitHub, Inc. [Terms](#) [Privacy](#) [Security](#) [Status](#) [Docs](#) [Contact GitHub](#) [Pricing](#) [API](#) [Training](#)

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 catkin_make_isolated	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 ФИО и теоретический конспект лекции. Сохраняем набранный текст через кнопку «Commit changes».

The screenshot shows a GitHub repository editor for a repository named 'Py3_3' by 'RomanGorchakov'. The file being edited is 'README.md' in the 'main' branch. The editor interface includes a header with the repository name, a search bar, and navigation links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The main content area shows the README text, which includes a title, a description of operator overloading, and Python code examples for class A and class B(A). The bottom of the editor has a 'Commit changes...' button.

```

1  Горчаков Роман Владимирович. Вариант 2
2  # Лабораторная работа 4.2. Перегрузка операторов в языке Python
3
4  Перегрузка операторов – это один из способов реализации полиморфизма, когда мы можем задать свою реализацию какого-либо метода в своём классе. Например, у нас есть два класса:
5
6  > class A:
7  >> def go(self):
8  >>> print('Go, A!')
9
10 > class B(A):
11 >> def go(self, name):
12 >>> print('Go, {}!'.format(name))
13
14 В данном примере класс B наследует класс A, но переопределяет метод go, поэтому он имеет мало общего с аналогичным методом класса A.
15
16 Однако в Python имеются методы, которые, как правило, не вызываются напрямую, а вызываются встроенными функциями или операторами. Например, метод __init__ перегружает конструктор класса. Конструктор – это создание экземпляра класса.
17
18 > class A:
19 >> def __init__(self, name):
20 >>> self.name = name

```

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. Устанавливаем библиотеки isort, black и flake8 и создаём файлы .pre-commit-config.yaml и environment.yml.

```
• @RomanGorchakov →/workspaces/Py3_3 (main) $ git checkout -b develop
Switched to a new branch 'develop'
• @RomanGorchakov →/workspaces/Py3_3 (develop) $ git branch feature_branch
• @RomanGorchakov →/workspaces/Py3_3 (develop) $ git branch release/1.0.0
• @RomanGorchakov →/workspaces/Py3_3 (develop) $ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
• @RomanGorchakov →/workspaces/Py3_3 (main) $ git branch hotfix
• @RomanGorchakov →/workspaces/Py3_3 (main) $ git checkout develop
Switched to branch 'develop'
○ @RomanGorchakov →/workspaces/Py3_3 (develop) $
```

```
Collecting black
  Downloading black-24.10.0-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.manylinux_2_28_x86_64.whl.metadata (79 kB)
Collecting click>=8.0.0 (from black)
  Downloading click-8.1.7-py3-none-any.whl.metadata (3.0 kB)
Collecting mypy_extensions>=0.4.3 (from black)
  Downloading mypy_extensions-1.0.0-py3-none-any.whl.metadata (1.1 kB)
Requirement already satisfied: packaging>=22.0 in /home/codespace/.local/lib/python3.12/site-packages (from black) (24.1)
Collecting pathspec>=0.9.0 (from black)
  Downloading pathspec-0.12.1-py3-none-any.whl.metadata (21 kB)
Requirement already satisfied: platformdirs>=2 in /home/codespace/.local/lib/python3.12/site-packages (from black) (4.3.6)
Downloading black-24.10.0-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.manylinux_2_28_x86_64.whl (1.8 MB)
1.8/1.8 MB 19.8 MB/s eta 0:00:00
Downloading click-8.1.7-py3-none-any.whl (97 kB)
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-24.10.0 click-8.1.7 mypy_extensions-1.0.0 pathspec-0.12.1
• @RomanGorchakov →/workspaces/Py3_3 (develop) $ pip install flake8
Collecting flake8
  Downloading flake8-7.1.1-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<2.13.0,>=2.12.0 (from flake8)
  Downloading pycodestyle-2.12.1-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 flake8-7.1.1-py2.py3-none-any.whl (57 kB)
Downloading mccabe-0.7.0-py2.py3-none-any.whl (7.3 kB)
Downloading pycodestyle-2.12.1-py2.py3-none-any.whl (31 kB)
Downloading pyflakes-3.2.0-py2.py3-none-any.whl (62 kB)
Installing collected packages: pyflakes, pycodestyle, mccabe, flake8
Successfully installed flake8-7.1.1 mccabe-0.7.0 pycodestyle-2.12.1 pyflakes-3.2.0
• @RomanGorchakov →/workspaces/Py3_3 (develop) $ pre-commit sample-config > .pre-commit-config.yaml
• @RomanGorchakov →/workspaces/Py3_3 (develop) $ conda env export > environment.yml
○ @RomanGorchakov →/workspaces/Py3_3 (develop) $
```

5. Создаём файл «example.py», в котором нужно изменить класс Rational из примера 1 лабораторной работы 4.1, используя перегрузку операторов.

```
Командная строка
Microsoft Windows [Version 10.0.22631.4317]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\Admin>cd C:\Users\Admin\Desktop\3 курс\Объектно-ориентированное программирование\3\Пример

C:\Users\Admin\Desktop\3 курс\Объектно-ориентированное программирование\3\Пример>python example.py
r1 = 3 / 4
r2 = 5 / 6
r1 + r2 = 19 / 12
r1 - r2 = -1 / 12
r1 * r2 = 5 / 8
r1 / r2 = 9 / 10
r1 == r2: False
r1 != r2: True
r1 > r2: False
r1 < r2: True
r1 >= r2: False
r1 <= r2: True

C:\Users\Admin\Desktop\3 курс\Объектно-ориентированное программирование\3\Пример>
```

6. Создаём файл «individual1.py», в котором нужно создать класс с двумя дробными числами с реализацией возведения первого числа в степень, значение которой равно второму числу, максимально задействовав имеющиеся в Python средства перегрузки операторов.

```
Командная строка
C:\Users\Admin\Desktop\3 курс\Объектно-ориентированное программирование\3\Индивидуальные задания\Задание 1>python individual1.py
3.0 ^ 3.0 = 27.0
Введите первое число: 5
Введите второе число: 2
5.0 ^ 2.0 = 25.0
True
False
False
25.0 + 27.0 = 52.0
27.0 * 25.0 = 675.0
3.510777123127852 ^ 6.286836151420429 = 2684.484684069497
False
True
False
2684.484684069497 - 27.0 = 2657.484684069497
2684.484684069497 / 25.0 = 107.37938736277987

C:\Users\Admin\Desktop\3 курс\Объектно-ориентированное программирование\3\Индивидуальные задания\Задание 1>
```

7. Создаём файл «individual2.py», в котором нужно создать класс для работы с беззнаковыми целыми десятичными числами, используя для представления числа список из 100 элементов типа int, каждый из которых является десятичной цифрой. Младшая цифра имеет меньший индекс (единицы – в нулевом элементе списка). Реальный размер списка задаётся как аргумент

конструктора инициализации. Реализуем арифметические операции, аналогичные встроенным для целых чисел, и операции сравнения.

```
Командная строка
C:\Users\Admin\Desktop\3 курс\Объектно-ориентированное программирование\3\Индивидуальные задания\Задание 2>python individual2.py
unsigned int 4 = 8589934596
Введите число: 40
Введите размер числа: 10
unsigned int 40 = 8589934593
True
True
True
4294967336 + 4294967300 = 4294967340
4294967300 * 4294967336 = 4294967456
unsigned int 82 = 8589934674
False
True
False
4294967378 - 4294967300 = 4294967378
4294967378 / 4294967336 = 9.778887033462524e-09
C:\Users\Admin\Desktop\3 курс\Объектно-ориентированное программирование\3\Индивидуальные задания\Задание 2>
```

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

```
create mode 100644 "\320\237\321\200\320\270\320\274\320\265\321\200\example.py"
@RomanGorchakov →/workspaces/Py3_3 (develop) $ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
@RomanGorchakov →/workspaces/Py3_3 (main) $ git merge develop
Updating 461099c..a2db2e4
Fast-forward
 .pre-commit-config.yaml | 40 +++++
 environment.yml          | 80 +++++
 .../320\227\320\260\320\264\320\260\320\275\320\270\320\265 1/individual1.py" | 135 +++++
 .../320\227\320\260\320\264\320\260\320\275\320\270\320\265 2/individual2.py" | 144 +++++
 .../320\2404.1 \320\223\320\276\321\200\321\207\320\260\320\272\320\276\320\262\320\240\320\222.pdf" | Bin 0 -> 643629 bytes
 "\320\237\321\200\320\270\320\274\320\265\321\200\example.py" | 191 +++++
 6 files changed, 590 insertions(+)
create mode 100644 .pre-commit-config.yaml
create mode 100644 environment.yml
create mode 100644 "\320\230\320\275\320\264\320\270\320\262\320\270\320\264\321\203\320\260\320\273\321\214\320\275\321\213\320\265 \320\267\320\260\320\264\320\260\320\275\320\270\321\217\320\227\320\260\320\264\320\260\320\275\320\270\320\265 1/individual1.py"
create mode 100644 "\320\230\320\275\320\264\320\270\320\262\320\270\320\264\321\203\320\260\320\273\321\214\320\275\321\213\320\265 \320\267\320\260\320\264\320\260\320\275\320\270\321\217\320\227\320\260\320\264\320\260\320\275\320\270\320\265 2/individual2.py"
create mode 100644 "\320\236\321\202\321\207\321\221\321\202\320\233\320\2404.1 \320\223\320\276\321\200\321\207\320\260\320\272\320\276\320\262\320\240\320\222.pdf"
create mode 100644 "\320\237\321\200\320\270\320\274\320\265\321\200\example.py"
@RomanGorchakov →/workspaces/Py3_3 (main) $ git push -u
Enumerating objects: 14, done.
Counting objects: 100% (14/14), done.
Delta compression using up to 2 threads
Compressing objects: 100% (10/10), done.
Writing objects: 100% (13/13), 590.34 KiB | 21.08 MiB/s, done.
Total 13 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/RomanGorchakov/Py3_3
 461099c..a2db2e4 main -> main
branch 'main' set up to track 'origin/main'.
@RomanGorchakov →/workspaces/Py3_3 (main) $
```

Py3_3 Public

Help us improve GitHub Codespaces
Tell us how to make GitHub Codespaces work better for you with three quick questions. [Give feedback](#)

main 1 Branch Tags Go to file Add file Code

RomanGorchakov Operatoors a2db2e4 · now 5 Commits

Индивидуальные задания	Operatoors	now
Отчёт	Operatoors	now
Пример	Operatoors	now
.gitignore	Create .gitignore	last week
.pre-commit-config.yaml	Operatoors	now
LICENSE	Create LICENSE	last week
README.md	Update README.md	8 hours ago
environment.yml	Operatoors	now

README MIT license

Горчаков Роман Владимирович. Вариант 2

Лабораторная работа 4.2. Перегрузка операторов в языке Python

Перегрузка операторов — это один из способов реализации полиморфизма, когда мы можем задать свою реализацию какого-либо метода в своём классе. Например, у нас есть два класса:

About
No description, website, or topics provided.
Readme
MIT license
Activity
0 stars
1 watching
0 forks

Releases
No releases published
[Create a new release](#)

Packages
No packages published
[Publish your first package](#)

Languages
Python 100.0%

Suggested workflows
Based on your tech stack
PyLint [Configure](#)

Контрольные вопросы

1. Какие средства существуют в Python для перегрузки операций?

В Python для перегрузки операций используются специальные методы в классах. Имена таких методов включают двойное подчёркивание спереди и сзади. Также для реализации перегрузки можно использовать функции из модуля `functools`, например, `singledispatch` или `singledispatchmethod`.

2. Какие существуют методы для перегрузки арифметических операций и операций отношения в языке Python?

- 1) `__add__(self, other)` – сложение. `x + y` вызывает `x.__add__(y)`.
- 2) `__sub__(self, other)` – вычитание (`x - y`).
- 3) `__mul__(self, other)` – умножение (`x * y`).
- 4) `__truediv__(self, other)` – деление (`x / y`).
- 5) `__floordiv__(self, other)` – целочисленное деление (`x // y`).

- 6) `__mod__(self, other)` – остаток от деления ($x \% y$).
- 7) `__divmod__(self, other)` – частное и остаток (`divmod(x, y)`).
- 8) `__pow__(self, other[, modulo])` – возведение в степень ($x ** y$, `pow(x, y[, modulo])`).
- 9) `__lshift__(self, other)` – битовый сдвиг влево ($x << y$).
- 10) `__rshift__(self, other)` – битовый сдвиг вправо ($x >> y$).
- 11) `__and__(self, other)` – битовое И ($x \& y$).
- 12) `__xor__(self, other)` – битовое ИСКЛЮЧАЮЩЕЕ ИЛИ ($x \wedge y$).
- 13) `__or__(self, other)` – битовое ИЛИ ($x | y$).
- 14) `__iadd__(self, other)` – $+=$.
- 15) `__isub__(self, other)` – $-=$.
- 16) `__imul__(self, other)` – $*=$.
- 17) `__itruediv__(self, other)` – $/=$.
- 18) `__ifloordiv__(self, other)` – $//=$.
- 19) `__imod__(self, other)` – $\%=$.
- 20) `__ipow__(self, other[, modulo])` – $**=$.
- 21) `__ilshift__(self, other)` – $<<=$.
- 22) `__irshift__(self, other)` – $>>=$.
- 23) `__iand__(self, other)` – $\&=$.
- 24) `__ixor__(self, other)` – $\wedge=$.
- 25) `__ior__(self, other)` – $|=$.
- 26) `__neg__(self)` – унарный $-$.
- 27) `__pos__(self)` – унарный $+$.
- 28) `__abs__(self)` – модуль (`abs()`).
- 29) `__invert__(self)` – инверсия (\sim).
- 30) `__complex__(self)` – приведение к `complex`.
- 31) `__int__(self)` – приведение к `int`.
- 32) `__float__(self)` – приведение к `float`.
- 33) `__round__(self[, n])` – округление.

34) `__enter__(self)` , `__exit__(self, exc_type, exc_value, traceback)` – реализации менеджеров контекста, используемый оператором `with`.

3. В каких случаях будут вызваны следующие методы: `__add__`, `__iadd__` и `__radd__`? Приведите примеры.

Метод `add` вызывается, когда экземпляр класса стоит слева от сложения. Метод `radd` вызывается, когда экземпляр класса находится справа от сложения, а слева от него не является экземпляром класса. Метод `iadd` вызывается для реализации оператора `+=`. Он позволяет изменять объект, не создавая новые экземпляры класса.

4. Для каких целей предназначен метод `__new__`? Чем он отличается от метода `__init__`?

Метод `__new__` вызывается при создании нового экземпляра класса. Он отвечает за выделение памяти под новый объект и возвращает этот объект. `__new__` является статическим методом и должен возвращать экземпляр класса. С другой стороны, метод `__init__` вызывается после `__new__`, когда объект уже создан. Он отвечает за инициализацию этого объекта, то есть задаёт начальные значения его атрибутов.

5. Чем отличаются методы `__str__` и `__repr__`?

Метод `__str__` возвращает строковое представление объекта. Метод `__repr__` вызывается встроенной функцией `repr`; он возвращает «сырые» данные, используемые для внутреннего представления в Python.