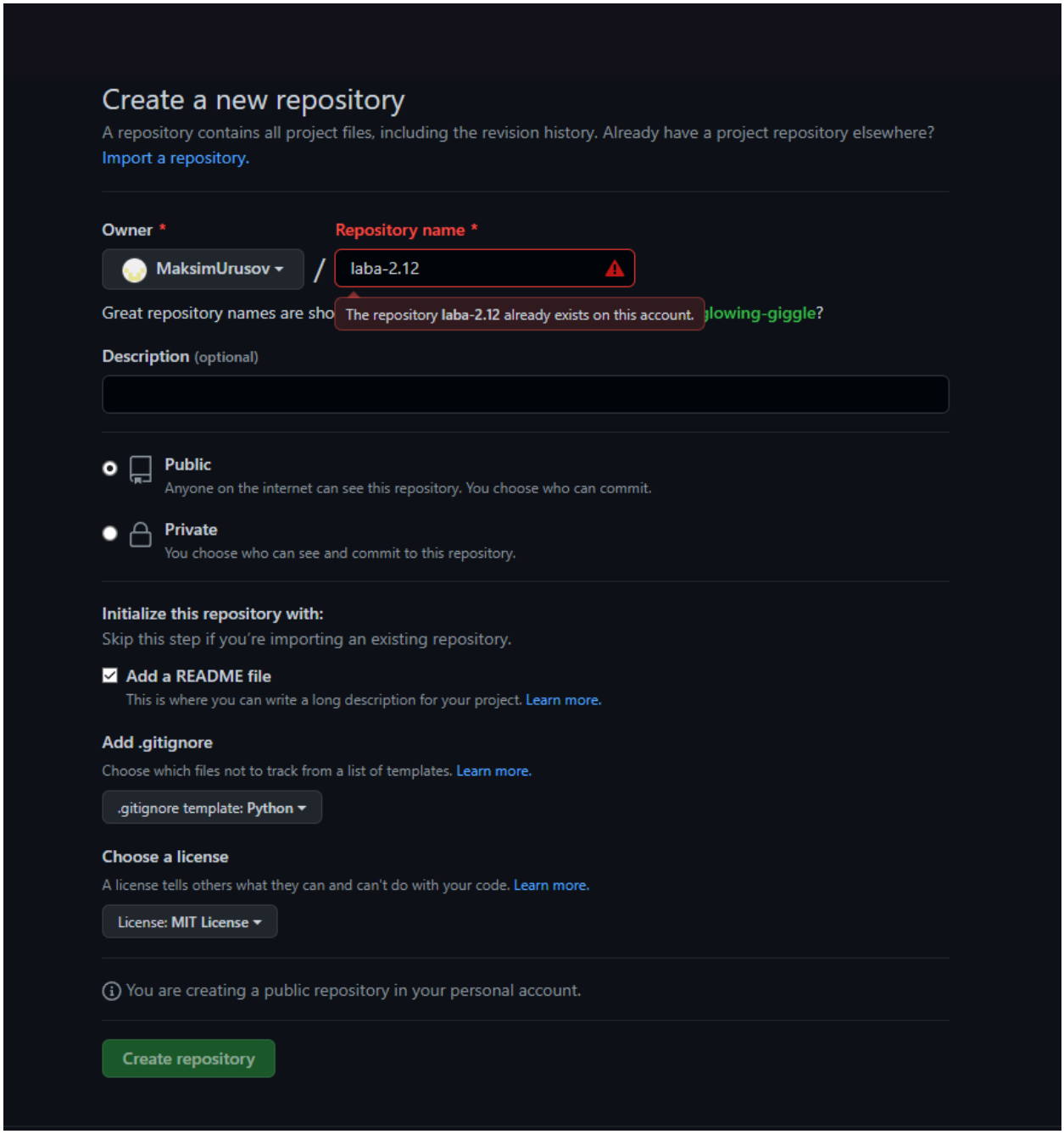
Лабораторная работа №6

Урусов Максим Андреевич

ИВТ-б-о-21-1

Цель: приобретение навыков по работе с функциями при написании программ с помощью языка программирования Python версии 3.x.

1. Создал общедоступный репозиторий на GitHub с MIT



2. Выполнил клонирование созданного репозитория.

```
D:\Program Files\Git>git clone https://github.com/MaksimUrusov/Laba-2.12.git
Cloning into 'Laba-2.12'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), done.
```

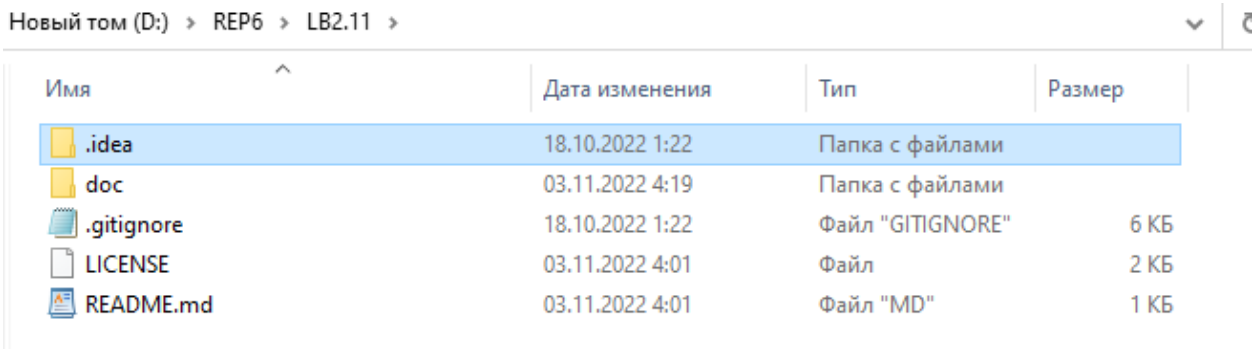
3. Дополнил файл .gitignore необходимыми правилами для работы с IDE PyCharm.

```
# Created by https://www.toptal.com/developers/gitignore/api/python,pycharm
# Edit at https://www.toptal.com/developers/gitignore?templates=python,pycharm

### PyCharm ###
# Covers JetBrains IDEs: IntelliJ, RubyMine, PhpStorm, AppCode, PyCharm, CLion, Android Studio, WebStorm and Rider
# Reference: https://intellij-support.jetbrains.com/hc/en-us/articles/206544839

# User-specific stuff
.idea/**/workspace.xml
.idea/**/tasks.xml
.idea/**/usage.statistics.xml
.idea/**/dictionaries
.idea/**/shelf

# AWS User-specific
.idea/**/aws.xml

# Generated files
.idea/**/contentModel.xml

# Sensitive or high-churn files
.idea/**/dataSources/
.idea/**/dataSources.ids
.idea/**/dataSources.local.xml
.idea/**/sqlDataSources.xml
.idea/**/dynamic.xml
.idea/**/uiDesigner.xml
.idea/**/dbnavigator.xml

# Gradle
.idea/**/gradle.xml
.idea/**/libraries

# Gradle and Maven with auto-import
# When using Gradle or Maven with auto-import, you should exclude module files,
# since they will be recreated, and may cause churn.  Uncomment if using
# auto-import.
```

4. Организовал репазиторий в соответствии с моделью ветвления git-flow.

```
D:\Program Files\Git>cd laba-2.12

D:\Program Files\Git\Laba-2.12>git flow init

Which branch should be used for bringing forth production releases?
   - main
Branch name for production releases: [main]
Branch name for "next release" development: [develop]
fatal: '?[C' is not a valid branch name
error: pathspec '?[C' did not match any file(s) known to git
Fatal: Could not check out branch ' '.
```

5. Создал проэкт пайчарм



6. Проработал примеры ла

```python
import math


new *
def decorator(func):
    new *
    def wrapper(r):
        print('Площадь круга равна = {:.3f}'.format(func(r)))
    return wrapper


new *
@decorator
def circle(r):
    s = math.pi * pow(r, 2)
    return s


if __name__ == "__main__":
    circle(11)
```

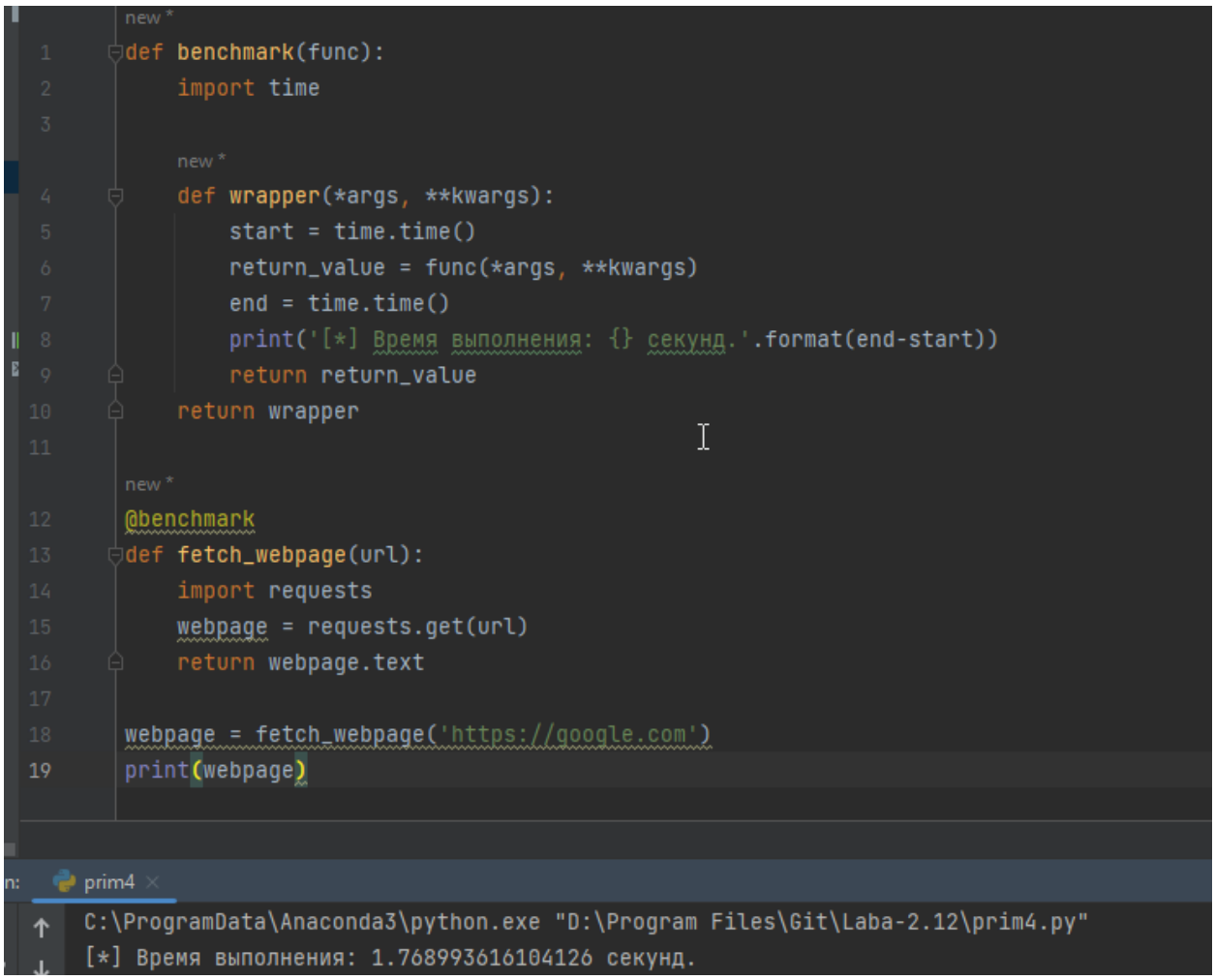
```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

new *
def hello_world():
    print('Hello world!')


new *
def higher_order(func):
    print('Получена функция {} в качестве аргумента'.format(func))
    func()
    return func

if __name__ == '__main__':

    higher_order(hello_world)
```

```
if __name__ == '__main__'
prim3
C:\ProgramData\Anaconda3\python.exe "D:\Program Files\Git\Laba-2.12\prim3.py"
Получена функция <function hello_world at 0x000001AA5951F0D0> в качестве аргумента
Hello world!

Process finished with exit code 0
```

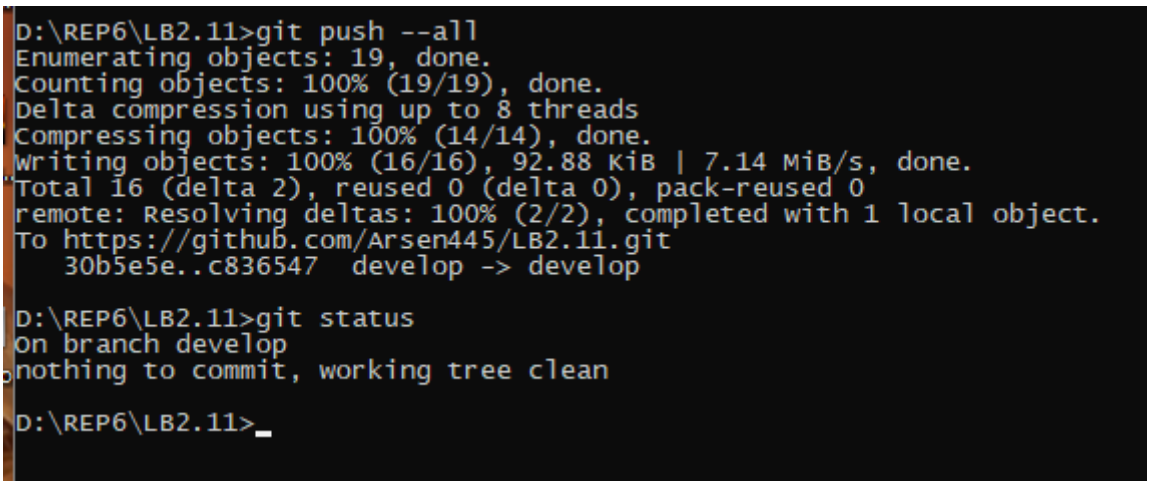
бораторной работы.

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

new *
def hello_world():
    print("Hello world!")


if __name__ == '__main__':

    a = type(hello_world)
    print (a)
```

```
_name__ == '__main__'
prim1
C:\ProgramData\Anaconda3\python.exe "D:\Program Files\Git\Laba-2.12\prim1.py"
<class 'function'>

Process finished with exit code 0
```



7. Зафиксируйте сделанные изменения в репозитории.(после создания веток не запушил, поэтому не работало)



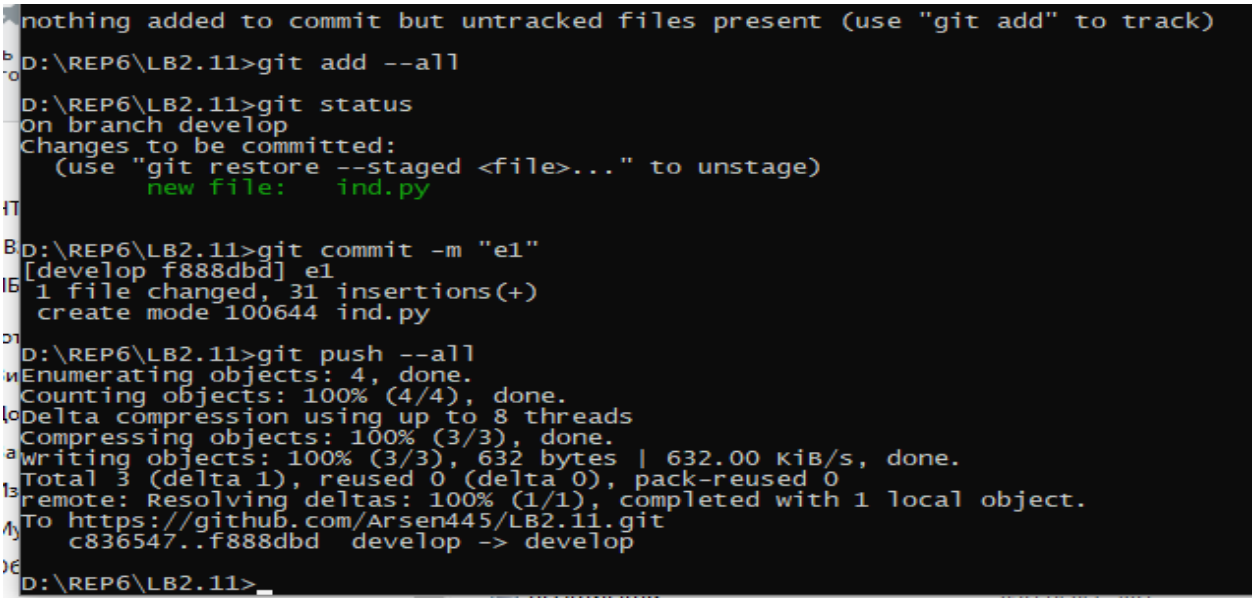
8. Решите ивидуальное задание.

   Используя замыкания функций, объявите внутреннюю функцию, которая на основе двух параметров вычисляет площадь фигуры. Какой именно фигуры: треугольника или прямоугольника, определяется параметром type внешней функции. Если type принимает значение 0, то вычисляется площадь треугольника, а иначе – прямоугольника. По умолчанию параметр type должен быть равен 0. Вычисленное значение должно возвращаться внутренней функцией. Вызовите внутреннюю функцию замыкания и отобразите на экране результат ее работы.

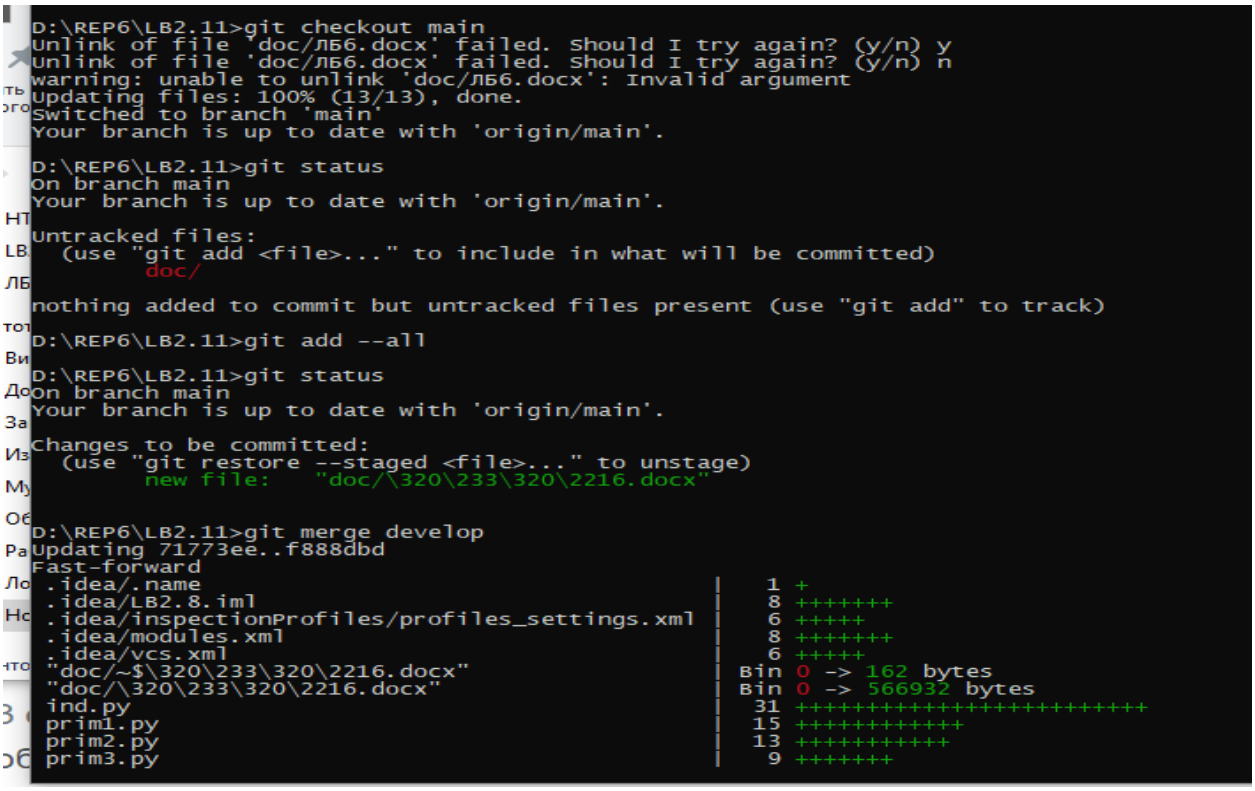
```python
import math


new *
def decorator(func):
    new *
    def wrapper(r):
        print('Площадь круга равна = {:.3f}'.format(func(r)))
    return wrapper


new *
@decorator
def circle(r):
    s = math.pi * pow(r, 2)
    return s


if __name__ == "__main__":
    circle(11)
```

```
C:\ProgramData\Anaconda3\python.exe "D:\Program Files\Git\Laba-2.12\ind.py"
Площадь круга равна = 380.133

Process finished with exit code 0
```

9. Зафиксируйте сделанные изменения в репозитории.



10. Выполните слияние ветки для разработки с веткой main/master.



Контрольные вопросы:

1. **Что такое замыкание?**

“замыкание (closure) в программировании — это функция, в теле которой присутствуют ссылки на переменные, объявленные вне тела этой функции в окружающем коде и не являющиеся ее параметрами.”

2. **Как реализованы замыкания в языке программирования Python?**

3. **Что подразумевает под собой область видимости Local?**

   Эту область видимости имеют переменные, которые создаются и используются внутри функций.

4. **Что подразумевает под собой область видимости Enclosing?**

   Суть данной области видимости в том, что внутри функции могут быть вложенные функции и локальные переменные, так вот локальная переменная функции для ее вложенной функции находится в enclosing области видимости.

5. **Что подразумевает под собой область видимости Global?**

   Переменные области видимости global – это глобальные переменные уровня модуля (модуль – это файл с расширением .py)

6. **Что подразумевает под собой область видимости Build-in?**

   Уровень Python интерпретатора. В рамках этой области видимости находятся функции open, len и т. п., также туда входят исключения. Эти сущности доступны в любом модуле Python и не требуют предварительного импорта. Built-in – это максимально широкая область видимости

7. **Как использовать замыкания в языке программирования Python?**

```
>>> def fun1(a):
        x = a * 3
        def fun2(b):
            nonlocal x
            return b + x
        return fun2

>>> test_fun = fun1(4)

>>> test_fun(7)
19
```

8. **Как замыкания могут быть использованы для построения иерархических данных?**

```
>>> tpl = lambda a, b: (a, b)
```

Если мы передадим в качестве аргументов числа, то, получим простой кортеж.

```
>>> a = tpl(1, 2)
>>> a
(1, 2)
```

Эту операцию можно производить не только над числами, но и над сущностями, ей же и порожденными.

```
>>> b = tpl(3, a)
>>> b
(3, (1, 2))

>>> c = tpl(a, b)
>>> c
((1, 2), (3, (1, 2)))
```