

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение  
высшего образования  
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития  
Кафедра инфокоммуникаций

**«Тестирование в Python [unittest]»**

**ОТЧЕТ**  
**по лабораторной работе №22**  
**дисциплины**  
**«Основы программной инженерии»**

Выполнила:

Кувшин Ирина Анатольевна  
2 курс, группа ПИЖ-б-о-21-1,  
011.03.04 «Программная инженерия»,  
направленность (профиль) «Разработка  
и сопровождение программного  
обеспечения», очная форма обучения

---

(подпись)

Проверил:

---

(подпись)

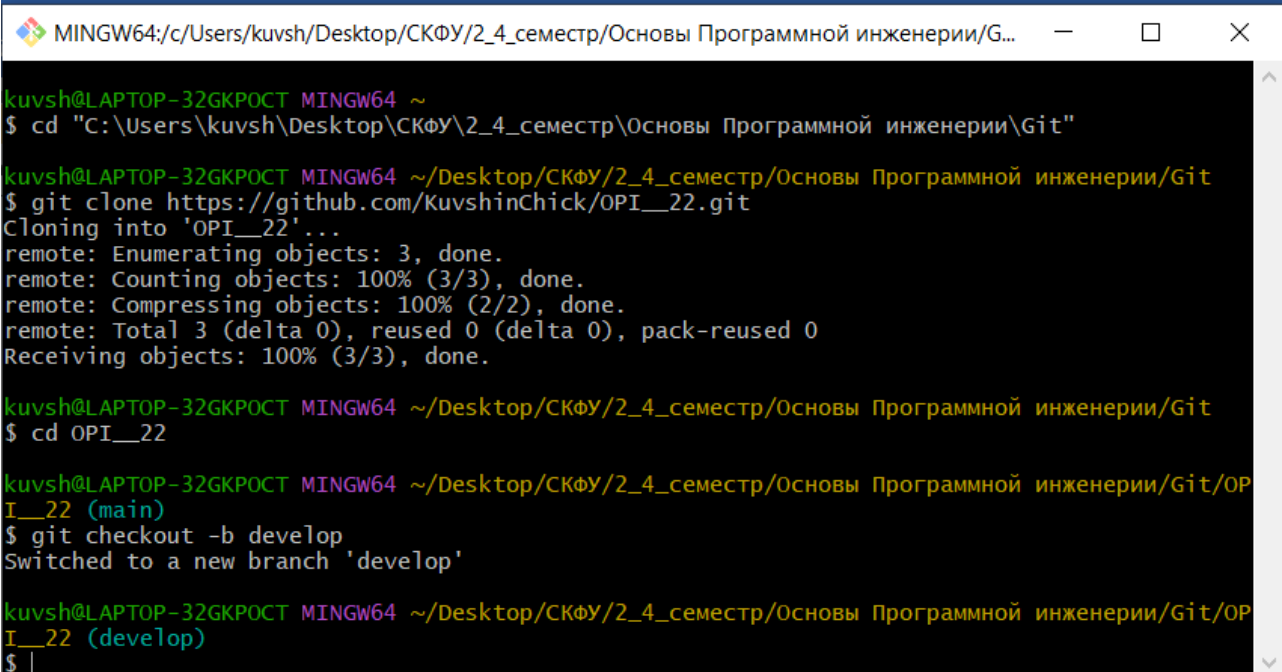
Отчет защищен с оценкой \_\_\_\_\_ Дата защиты \_\_\_\_\_

Ставрополь, 2023 г.

**Цель работы:** приобретение навыков написания автоматизированных тестов на языке программирования Python версии 3.x.

**Ход работы:**

1. Изучить теоретический материал работы.
2. Проработайте примеры лабораторной работы.
3. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python.
4. Выполните клонирование созданного репозитория.
5. Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm.
6. Организуйте свой репозиторий в соответствии с моделью ветвления git-flow.
7. Создайте проект PyCharm в папке репозитория.



```
MINGW64/c:/Users/kuvsh/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/G...
kuvsh@LAPTOP-32GKPOCT MINGW64 ~
$ cd "C:\Users\kuvsh\Desktop\СКФУ\2_4_семестр\Основы Программной инженерии\Git"

kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git
$ git clone https://github.com/KuvshinChick/OPI__22.git
Cloning into 'OPI__22'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.

kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git
$ cd OPI__22

kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git/OPI__22 (main)
$ git checkout -b develop
Switched to a new branch 'develop'

kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git/OPI__22 (develop)
$ |
```

Рисунок 22.1 – Клонирование репозитория и создание ветки develop

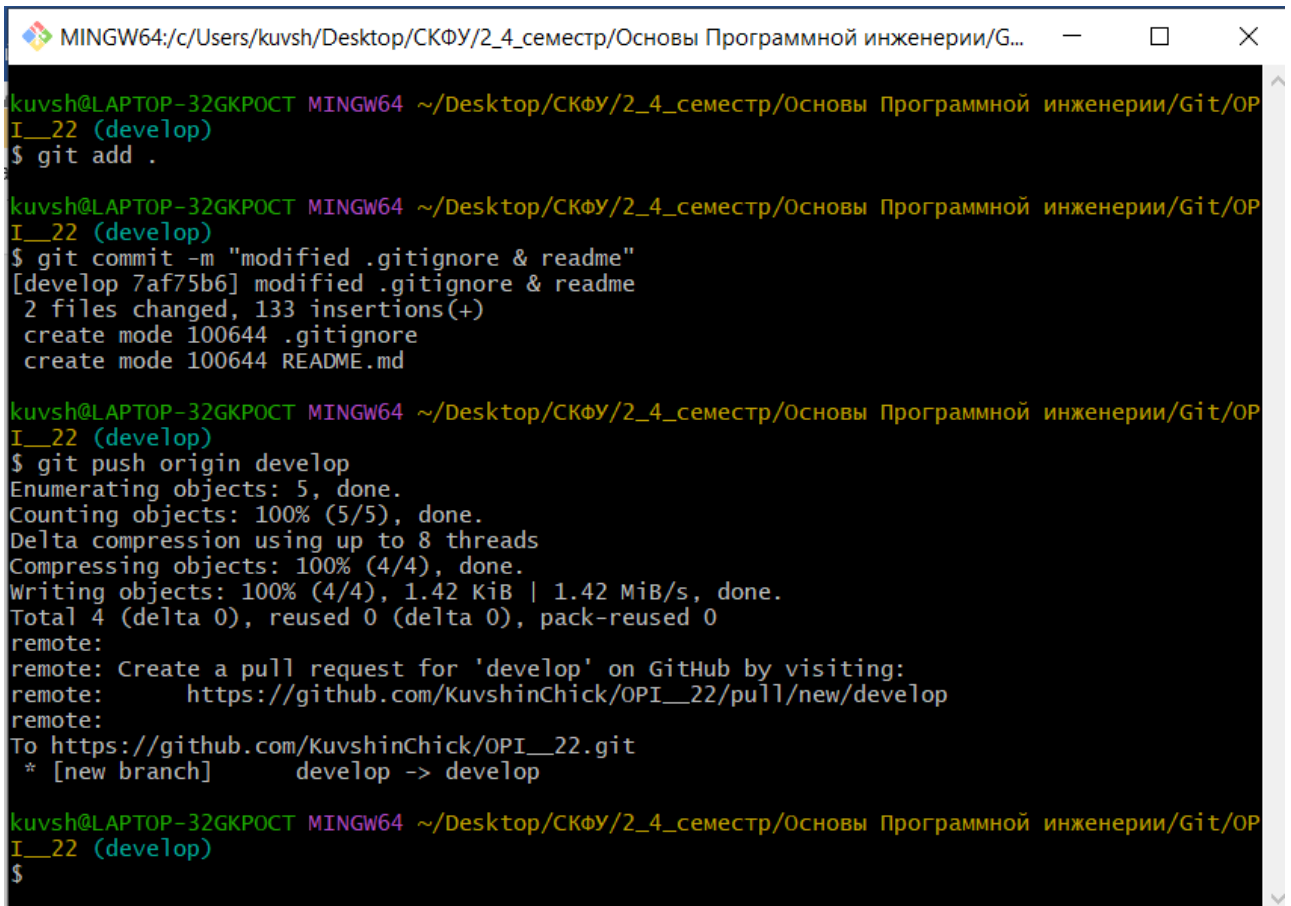
A screenshot of a terminal window with a black background and green and white text. The window title is 'MINGW64:/c:/Users/kuvsh/Desktop/СКФУ/2\_4\_семестр/Основы Программной инженерии/G...'. The user is 'kuvsh@LAPTOP-32GKPOCT' and the shell is 'MINGW64'. The current directory is '~/Desktop/СКФУ/2\_4\_семестр/Основы Программной инженерии/Git/OPI\_\_22 (develop)'. The commands and output are as follows:  
\$ git add .  
\$ git commit -m "modified .gitignore & readme"  
[develop 7af75b6] modified .gitignore & readme  
2 files changed, 133 insertions(+)  
create mode 100644 .gitignore  
create mode 100644 README.md  
\$ git push origin develop  
Enumerating objects: 5, done.  
Counting objects: 100% (5/5), done.  
Delta compression using up to 8 threads  
Compressing objects: 100% (4/4), done.  
Writing objects: 100% (4/4), 1.42 KiB | 1.42 MiB/s, done.  
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0  
remote:  
remote: Create a pull request for 'develop' on GitHub by visiting:  
remote: https://github.com/KuvshinChick/OPI\_\_22/pull/new/develop  
remote:  
To https://github.com/KuvshinChick/OPI\_\_22.git  
\* [new branch] develop -> develop  
\$

Рисунок 22.2 – Обновление .gitignore и readme

8. Выполните индивидуальное задание. Приведите в отчете скриншоты работы программы решения индивидуального задания.
9. Зафиксируйте сделанные изменения в репозитории.
10. Выполните слияние ветки для разработки с веткой main (master).
11. Отправьте сделанные изменения на сервер GitHub.

### Индивидуальное задание

Для индивидуального задания лабораторной работы 2.21 добавьте тесты с использованием модуля unittest, проверяющие операции по работе с базой данных.

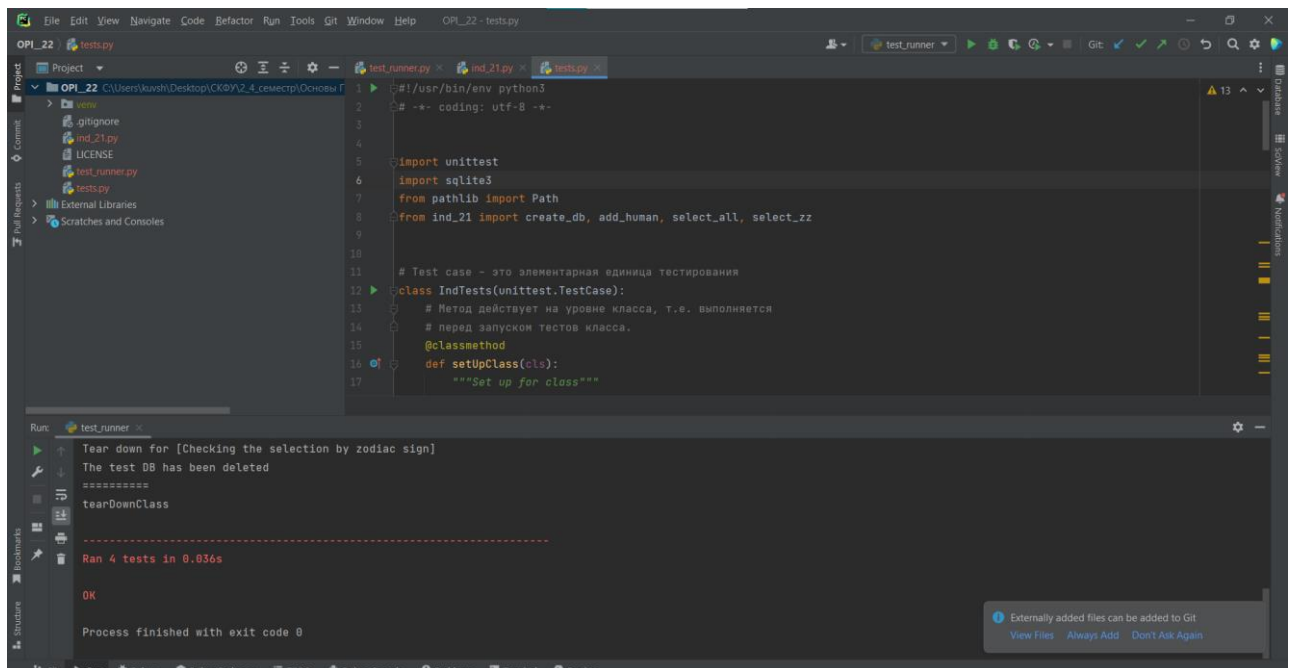


Рисунок 22.3 – Проработка программы

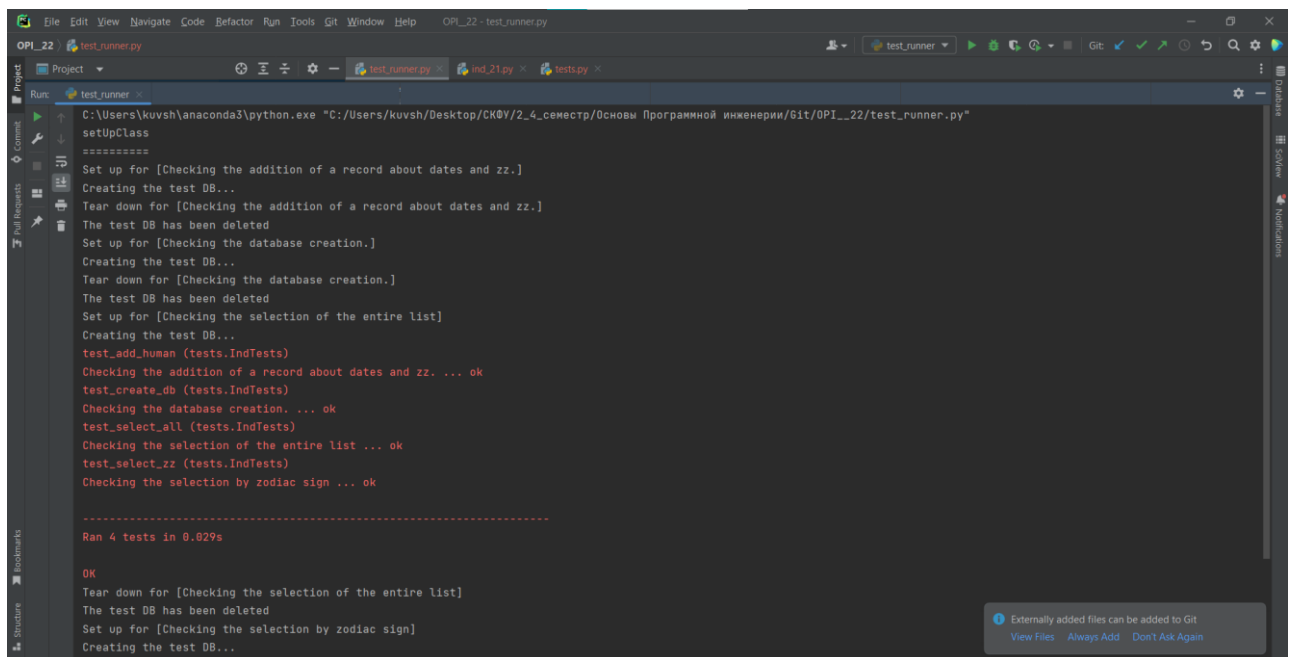


Рисунок 22.4 – Результат работы программы

## Контрольные вопросы

1. Для чего используется автономное тестирование?

Автономный тест – это автоматизированная часть кода, которая вызывает тестируемую единицу работы и затем проверяет некоторые предположения о единственном конечном результате этой единицы. В качестве тестируемой единицы, в данном случае, может выступать как отдельный метода (функция),

так и совокупность классов (или функций). Идея автономной единицы в том, что она представляет собой некоторую логически законченную сущность вашей программы. Автономное тестирование ещё называют модульным или unit-тестированием (unit-testing). Здесь и далее под словом тестирование будет пониматься именно автономное тестирование.

2. Какие фреймворки Python получили наибольшее распространение для решения задач автономного тестирования?

В мире Python существуют три framework'a, которые получили наибольшее распространение:

`unittest`

`nose`

`pytest`

3. Какие существуют основные структурные единицы модуля `unittest`?

=

## Основные структурные элементы unittest

*unittest* – это *framework* для тестирования в *Python*, который позволяет разрабатывать автономные тесты, собирать тесты в коллекции, обеспечивает независимость тестов от *framework'a* отчетов и т.д. Основными структурными элементами каркаса *unittest* являются:

### Test fixture

*Test fixture* – обеспечивает подготовку окружения для выполнения тестов, а также организацию мероприятий по их корректному завершению (например очистка ресурсов). Подготовка окружения может включать в себя создание баз данных, запуск необходим серверов и т.п.

### Test case

*Test case* – это элементарная единица тестирования, в рамках которой проверяется работа компонента тестируемой программы (метод, класс, поведение и т. п.). Для реализации этой сущности используется класс *TestCase*.

### Test suite

*Test suite* – это коллекция тестов, которая может в себя включать как отдельные *test case'ы* так и целые коллекции (т.е. можно создавать коллекции коллекций). Коллекции используются с целью объединения тестов для совместного запуска.

### Test runner

*Test runner* – это компонент, который оркестрирует (координирует взаимодействие) запуск тестов и предоставляет пользователю результат их выполнения. *Test runner* может иметь графический интерфейс, текстовый интерфейс или возвращать какое-то заранее заданное значение, которое будет описывать результат прохождения тестов.

Вся работа по написанию тестов заключается в том, что мы разрабатываем отдельные тесты в рамках *test case'ов*, собираем их в модули и запускаем, если нужно объединить несколько *test case'ов*, для их совместного запуска, они помещаются в *test suite'ы*, которые помимо *test case'ов* могут содержать другие *test suite'ы*.

## 4. Какие существуют способы запуска тестов unittest?

## Интерфейс командной строки (CLI)

CLI позволяет запускать тесты из целого модуля, класса или даже обращаться к конкретному тесту.

Запуск всех тестов в модуле *utest\_calc.py*.

```
> python -m unittest test_calc.py
```

Запуск тестов из класса *CalcTest*.

```
> python -m unittest utest_calc.CalcTest
```

Запуск теста *test\_sub()*.

---

```
> python -m unittest utest_calc.CalcTest.test_sub
```

Как уже было сказано ранее, для вывода подробной информации необходимо добавить ключ *-v*.

```
> python -m unittest -v utest_calc.py
```

Если осуществить запуск без указания модуля с тестами, будет запущен Test Discovery.

```
> python -m unittest
```

Более подробно о Test Discovery будет рассказано в одной из следующих частей. Справку по ключам запуска можно получить из документации (<https://docs.python.org/3/library/unittest.html#command-line-options>).

## Графический интерфейс пользователя (GUI)

Для запуска и анализа результатов работы тестов можно использовать GUI. Список инструментов доступен по ссылке [https://wiki.python.org/moin/PythonTestingToolsTaxonomy#GUI\\_Testing\\_Tools](https://wiki.python.org/moin/PythonTestingToolsTaxonomy#GUI_Testing_Tools), но он далеко не полный. Для пример рассмотрим работу с Cricket (<https://github.com/pybee/cricket>). Для установки *Cricket* можно воспользоваться менеджером pip:

```
> pip install cricket
```

После этого на ваш компьютер будет установлен *cricket-unittest*.

Для запуска тестов в данном приложении, перейдите в каталог с вашим тестирующим кодом и в командной строке запустите *cricket-unittest*, для этого просто наберите это название и нажмите *Enter*.

```
> cricket-unittest
```

Приложение, при запуске, автоматически загрузит тесты.

## 5. Каково назначение класса TestCase?

Как уже было сказано – основным строительным элементом при написании тестов с использованием unittest является TestCase. Он представляет собой класс, который должен являться базовым для всех остальных классов, методы которых будут тестировать те или иные автономные единицы исходной программы.

## 6. Какие методы класса TestCase выполняются при запуске и завершении работы тестов?

К этим методам относятся:

`setUp()`

Метод вызывается перед запуском теста. Как правило, используется для подготовки окружения для теста.

`tearDown()`

Метод вызывается после завершения работы теста. Используется для “приборки” за тестом.

## 7. Какие методы класса TestCase используются для проверки условий и генерации ошибок?



Метод	Описание
<code>assertEqual(a, b)</code>	<code>a == b</code>
<code>assertNotEqual(a, b)</code>	<code>a != b</code>
<code>assertTrue(x)</code>	<code>bool(x) is True</code>
<code>assertFalse(x)</code>	<code>bool(x) is False</code>
<code>assertIs(a, b)</code>	<code>a is b</code>
<code>assertIsNot(a, b)</code>	<code>a is not b</code>
<code>assertIsNone(x)</code>	<code>x is None</code>
<code>assertIsNotNone(x)</code>	<code>x is not None</code>
<code>assertIn(a, b)</code>	<code>a in b</code>
<code>assertNotIn(a, b)</code>	<code>a not in b</code>
<code>assertIsInstance(a, b)</code>	<code>isinstance(a, b)</code>
<code>assertNotIsInstance(a, b)</code>	<code>not isinstance(a, b)</code>

Assert'ы для контроля выбрасываемых исключений и warning'ов:

Метод	Описание
<code>assertRaises(exc, fun, *args, **kwargs)</code>	Функция <code>fun(*args, **kwargs)</code> вызывает исключение <code>exc</code>
<code>assertRaisesRegex(exc, r, fun, *args, **kwargs)</code>	Функция <code>fun(*args, **kwargs)</code> вызывает исключение <code>exc</code> , сообщение которого совпадает с регулярным выражением <code>r</code>
<code>assertWarns(warn, fun, *args, **kwargs)</code>	Функция <code>fun(*args, **kwargs)</code> выдает сообщение <code>warn</code>
<code>assertWarnsRegex(warn, r, fun, *args, **kwargs)</code>	Функция <code>fun(*args, **kwargs)</code> выдает сообщение <code>warn</code> и оно совпадает с регулярным выражением <code>r</code>

8. Какие методы класса `TestCase` позволяют собирать информацию о самом тесте?

`countTestCases()`

Возвращает количество тестов в объекте класса-наследника от `TestCase`.

`id()`

Возвращает строковый идентификатор теста. Как правило это полное имя метода, включающее имя модуля и имя класса.

`shortDescription()`

Возвращает описание теста, которое представляет собой первую строку `docstring`'а метода, если его нет, то возвращает `None`.

9. Каково назначение класса `TestSuite`? Как осуществляется загрузка тестов?

Класс `TestSuite` используется для объединения тестов в группы, которые могут включать в себя как отдельные тесты так и заранее созданные группы. Помимо этого, `TestSuite` предоставляет интерфейс, позволяющий `TestRunner`'у, запускать тесты. Разберем более подробно методы класса `TestSuite`.

10. Каково назначение класса `TestResult`?

Класс `TestResult` используется для сбора информации о результатах прохождения тестов. Подробную информацию по атрибутам и классам этого метода можно найти в официальной документации (<https://docs.python.org/3/library/unittest.html#unittest.TestResult>).

11. Для чего может понадобиться пропуск отдельных тестов?

Пропуск отдельных тестов может понадобиться, например, если они зависят от функциональности, которая еще не реализована.

12. Как выполняется безусловный и условных пропуск тестов? Как выполнить пропуск класса тестов?

Исключим тест `test_add` из списка тестов. При попытке решить такую задачу, первое, что может прийти на ум – это удалить либо закомментировать данный тест. Но *unittest* предоставляет нам инструменты для удобного управления процессом пропуска тестов. Это может быть ещё полезно в том плане, что информацию о пропущенных тестах (их количестве) можно дополнительно получить через специальный *API*, предоставляемый классом *TestResult*. Для пропуска теста воспользуемся декоратором, который пишется перед тестом.

```
@unittest.skip(reason)
```

Для условного пропуска тестов применяются следующие декораторы:

```
@unittest.skipIf(condition, reason)
```

Тест будет пропущен, если условие (*condition*) истинно.

```
@unittest.skipUnless(condition, reason)
```

Тест будет пропущен если, условие (*condition*) не истинно.

Условный пропуск тестов можно использовать в ситуациях, когда те или иные тесты зависят от версии программы, например: в новой версии уже не поддерживается часть методов; или тесты могут быть платформозависимые, например: ряд тестов могут выполняться только под операционной системой *MS Windows*. Условие записывается в параметр *condition*, текстовое описание – в *reason*.

13. Самостоятельно изучить средства по поддержке тестов *unittest* в *PyCharm*. Приведите обобщенный алгоритм проведения тестирования с помощью *PyCharm*.

Для проведения тестирования с помощью *PyCharm* необходимо создать новый проект, добавить тестовый модуль, написать тесты и запустить их с помощью контекстного меню или специальной кнопки. Общий алгоритм проведения тестирования следующий: создание тестового сценария, запуск тестов, анализ результатов и корректировка кода программы.