

## HW #05: Asset Mock

---

<b>1. Описание задания</b>	<b>2</b>
1.1. Консольное приложение по работе с активами	2
1.2. Требования к реализации	3
<b>2. Рекомендации</b>	<b>6</b>
<b>3. Критерии оценивания</b>	<b>7</b>
<b>4. Инструкция по отправке задания</b>	<b>8</b>
<b>5. FAQ (часто задаваемые вопросы)</b>	<b>10</b>
<b>6. Дополнительные задания (не на оценку)</b>	<b>11</b>

---

## 1. Описание задания

В этом задании вам нужно настроить Mock (в enclosing namespace), который будет иметь динамическое поведение в зависимости от числа вызовов. Цель задания:

1. Научиться строить Mock'и;
2. Научиться пользоваться правилом LEGB.

### 1.1. Консольное приложение по работе с активами

Консольное приложение по работе с активами доступно на Github курса:

- [github:big-data-team/python-course/./asset\\_with\\_external\\_dependency.py](https://github.com/big-data-team/python-course/./asset_with_external_dependency.py)

Приложение предоставляет возможность оценить инвестиционную прибыль от актива за разные периоды времени (в годах). Пример использования:

```
$ python3 asset.py --filepath asset.txt --periods 1 2 5
```

Где asset.txt содержит информацию об активе:

```
<name> <capital> <interest>
```

- name - имя актива
- capital - стоимость актива (например в рублях)
- interest - годовая прибыльность актива, выраженная в процентах (0.1 - значит 10%)

Данное приложение в отличие от asset.py имеет дополнительную внешнюю зависимость ([cbr.py](#)), которое позволяет получить курс USD Центрального Банка России. Предположим, что данная функциональность предоставляется коллегами из соседнего департамента и к текущему моменту еще недоступна:

```
def get_usd_course():  
    raise NotImplementedError
```

Именно поэтому, вам придется воспользоваться Mock для проверки валидности реализации asset.py.



## 1.2. Требования к реализации

С целью наработки практических навыков по определению правильного namespace и указания порядка @patch, материалы занятий для copy-paste не выкладываются.

Задание 1. Для лучшего усвоения понимания namespace вам нужно замочать вызовы sleep и убедиться, что эти mock'и работают:

- ограничение на время на выполнения всех тестов - 1 секунда
- ожидаемый coverage - 100%

Библиотека для покрытия тестами (sleepy.py):

```
import time
from time import sleep

def sleep_add(x, y):
    sleep(3)
    z = x + y
    return z

def sleep_multiply(x, y):
    time.sleep(5)
    z = x * y
    return z
```

Задание 2. Задание ориентировано на наработку практических навыков и для запоминания правильного порядка указания @patch и mock объектов в определении тест-функции. Допустим, в библиотеку sleepy.py добавили функцию:

```
def deepest_sleep_function(x, y):
    z = sleep_add(x, y)
    w = sleep_multiply(x, y)
    outcome = z + w
    return outcome
```

Заполните правильно:



```
from unittest.mock import patch
from sleepy import deepest_sleep_function
```

```
##code##
```

```
def test_can_mock_all_sleep(mock_sleep_add, mock_sleep_multiply):
    outcome = deepest_sleep_function(1, 2)
    assert 5 == outcome

    mock_sleep_add.assert_called_once_with(3)
    mock_sleep_multiply.assert_called_once_with(5)
```

Задание 3. Наложите @patch на метод calculate\_revenue класса Asset, чтобы возвращать значение 100500.0 (float). Проверка функциональности будет проводиться с помощью pytest fixture capsys:

```
from unittest.mock import patch
import pytest
from asset import print_asset_revenue
```

```
@pytest.fixture
```

```
def asset_filepath(tmpdir):
    asset_fileio = tmpdir.join("asset.txt")
    asset_fileio.write("property 1000000 0.1")
    asset_filepath_ = asset_fileio.strpath
    return asset_filepath_
```

```
@patch("asset.Asset")
```

```
def test_asset_calculate_revenue_always_return_100500(mock_asset_class,
asset_filepath, capsys):
```

```
##code##
```

```
periods = [1, 2, 5, 10]
with open(asset_filepath) as asset_fin:
    print_asset_revenue(asset_fin, periods=periods)

captured = capsys.readouterr()
```



```
assert len( periods ) == len( captured.out.splitlines() )
for line in captured.out.splitlines():
    assert "100500" in line
```

Задание 4. Воспользуемся расширенной функциональностью Asset, где предоставляется возможность оценивать инвестиционную прибыльность продукта с учетом курса обмена валют. Реализуйте Mock таким образом, чтобы проходил следующий тест:

```
from unittest.mock import patch
import pytest
from asset import Asset

@patch("cbr.get_usd_course")
def test_can_mock_external_calls(mock_get_usd_course):
    ##code##

    asset_property = Asset(name="property", capital=10**6, interest=0.1)
    assert asset_property.calculate_revenue_from_usd(years=1) ==
pytest.approx(76.54 * 10**5, abs=0.01)
    assert asset_property.calculate_revenue_from_usd(years=1) ==
pytest.approx(77.44 * 10**5, abs=0.01)
    with pytest.raises(ConnectionError):
        asset_property.calculate_revenue_from_usd(years=1)
```

Задание 5. Ограниченный список возможных значений не всегда хорошо. Давайте добавим динамики в наше приложение и предоставим возможность получать каждый раз новое значение курса валют. Для усвоения правила LEGB добавляется обязательное требование: Mock и вспомогательные функции должны быть реализованы полностью внутри одной тест-функции. Реализуйте Mock таким образом, чтобы проходил следующий тест:

```
from unittest.mock import patch
import pytest
from asset import Asset
```



```
@patch("cbr.get_usd_course")
def test_can_mock_external_calls(mock_get_usd_course):
    ##code##

    asset_property = Asset(name="property", capital=10**6, interest=0.1)
    for iteration in range(##iteration_count##1):
        expected_revenue = (76.32 + 0.1 * iteration) * asset_property.capital
    * asset_property.interest
        calculated_revenue =
asset_property.calculate_revenue_from_usd(years=1)
        assert calculated_revenue == pytest.approx(expected_revenue,
abs=0.01), (
            f"incorrect calculated revenue at iteration {iteration}"
        )
```

Задание 6. С целью привития хорошей культуры написания кода в дополнение к уже изученным ранее PEP-257 и PEP-008 на этой неделе рекомендуется изучить Google Python Style Guide. Изучите рекомендации по хорошему написанию кода:

- <https://google.github.io/styleguide/pyguide.html>

У автора курса точно есть свои привычки и аргументы, которые расходятся с рекомендациями Google Python Style Guide. Предлагаем в чате курса обсудить с какими аргументами вы например не согласны, а что интересного и нового почерпнули из этого документа. Хештеги для удобства обсуждения и поиска в чате #google\_style\_guide.

## 2. Рекомендации

Официальная документация по работе с Mock:

- <https://docs.python.org/dev/library/unittest.mock.html>

В задании 1 вам нужно отправить полноценное решение (test-файл целиком). В заданиях 2-5, вам нужно отправить только часть кода (помеченного как ##code##), убрав лишние отступы слева и сохранив в соответствующем файле.

---

<sup>1</sup> Это значение будет известно только в момент тестирования, чтобы не хардкодить список возможных значений



Просьба обратить внимание, что в качестве решения задания 5 требуется отправить лишь строки до цикла по итерациям (то есть только блок `##code##`). При отправке задания на сам итератор никак не влияем/никак не моделируем.

Если в вашем коде используются "indentation", то необходимо делать это с помощью пробельных символов, а не знаков табуляции (**знаки табуляции запрещены**). Для выравнивания внутреннего тела функции / цикла и т.п. конструкций необходимо использовать 4 пробельных символа.



## 3. Критерии оценивания

Балл за задачу складывается из:

- **20%** - за задание #1
- **20%** - за задание #2
- **20%** - за задание #3
- **20%** - за задание #4
- **20%** - за задание #5

Discounts (скидки и другие акции):

- **100%** за плагиат в решениях (всем участникам процесса)
- **100%** за посылку решения после hard deadline
- **30%** за посылку решения в после soft deadline и до hard deadline
- **5%** за каждую посылку после 2й посылки в день (каждый день можно делать до 2х посылок без штрафа)

лучший балл с 1-й попытки: 100%

лучший балл со 2-й попытки: 100%

лучший балл с 3-й попытки: 95%

лучший балл с 4-й попытки: 90%



## 4. Инструкция по отправке задания

Оформление задания:

- Код задания (Short name): **HW05:Asset Mock**
- Выполненное ДЗ запакуйте в архив `PY-MADE-2021-Q4_<Surname>_<Name>_HW#.zip`, пример `--PY-MADE-2021-Q4_Dral_Alexey_HW05.zip`. (Проверяйте отсутствие пробелов и невидимых символов после копирования имени отсюда.<sup>2</sup>) Если ваше решение лежит в папке `my_solution_folder`, то для создания архива `hw.zip` на Linux и Mac OS выполните команду<sup>3</sup>:
  - `zip -r hw.zip my_solution_folder/*`
- На Windows 7/8/10: необходимо выделить все содержимое директории `my_solution_folder/` нажать правую кнопку мыши на одном из выделенных объектов, выбрать в открывшемся меню "Отправить >", затем "Сжатая ZIP-папка". Теперь можно переименовать архив.
- Решение задания должно содержаться в одной папке.
- Перед проверкой убедитесь, что дерево вашего архива выглядит так:
  - | `PY-MADE-2021-Q4_<Surname>_<Name>_HW05.zip`
  - | `---- test_<Surname>_<Name>_mock_sleep.py`<sup>4</sup>
  - | `---- task_<Surname>_<Name>_mock_deep_sleep.py`<sup>5</sup>
  - | `---- task_<Surname>_<Name>_asset_mock_revenue_100500.py`<sup>6</sup>
  - | `---- task_<Surname>_<Name>_asset_mock_usd_simple.py`<sup>7</sup>
  - | `---- task_<Surname>_<Name>_asset_mock_usd_enclosing.py`<sup>8</sup>
  - | `-----*.txt`<sup>9</sup>
  - При несовпадении дерева вашего архива с представленным деревом, ваше решение не будет возможным автоматически проверить, а значит, и оценить его.
- Для того, чтобы сдать задание, необходимо:
  - Зарегистрироваться и залогиниться в сервисе [Everest](#)
  - Перейти на страницу приложения: [MADE Python Grader](#)
  - Выбрать вкладку Submit Job (если отображается иная).
  - Выбрать в качестве "Task" значение: **HW05:Asset Mock**<sup>10</sup>

<sup>2</sup> Онлайн инструмент для проверки: <https://www.soscisurvey.de/tools/view-chars.php>

<sup>3</sup> Флаг `-r` значит, что будет совершен рекурсивный обход по структуре директории

<sup>4</sup> Решение задания #1

<sup>5</sup> Решение задания #2

<sup>6</sup> Решение задания #3

<sup>7</sup> Решение задания #4

<sup>8</sup> Решение задания #5

<sup>9</sup> Все необходимые тестовые данные нужно генерировать с помощью pytest fixture `tmpdir` и объектов в памяти Python

<sup>10</sup> Сервисный ID: `python.asset_mock`



- Загрузить в качестве "Task solution" файл с решением
- В качестве Access Token указать тот, который был выслан по почте
- **Перед отправкой задания**, оставьте, пожалуйста, отзыв о домашнем задании по ссылке: [https://rebrand.ly/pymade2021q4\\_feedback\\_hw](https://rebrand.ly/pymade2021q4_feedback_hw). Это позволит нам скорректировать учебную нагрузку по следующим заданиям (в зависимости от того, сколько часов уходит на решение ДЗ), а также ответить на интересующие вопросы.

**Внимание:** если до дедлайна остается меньше суток, и вы знаете (сами проверили или коллеги сообщили), что сдача решений сломана, обязательно сдайте свое решение, прислав нам ссылку на выполненное задание (Job) на почту с темой письма "Short name. ФИО.". Например: **"HW05:Asset Mock. Иванов Иван Иванович."** Таким образом, мы сможем увидеть какое решение у вас было до дедлайна и сможем его оценить. Пример ссылки:

- <https://everest.distcomp.org/jobs/67893456230000abc0123def>

Любые вопросы / комментарии / предложения пишите согласно [предложениям](#) на портале.

Всем удачи!



## 5. FAQ (часто задаваемые вопросы)

"You are not allowed to run this application", что делать?

Если Вы видите надпись "You are not allowed to run this application" во вкладке Submit Job в Everest, то на данный момент сдача закрыта (нет доступных для сдачи домашних заданий, по техническим причинам или другое). Попробуйте, пожалуйста, еще раз через некоторое время. Если Вы еще ни разу не сдавали, у коллег сдача работает, но Вы видите такое сообщение, сообщите нам об этом.

Grader показывает 0 или  $< 0$ , а отчет (Grading report) не помогает решить проблему

Ситуации:

- система оценивания показывает оценку (Grade)  $< 0$ , а отчет (Grading report) не помогает решить проблему. Пример: в случае неправильно указанного access token система вернет -401 и информацию о том, что его нужно поправить;
- система показывает 0 и в отчете (Grading report) не указано, какие тесты не пройдены. Пример: вы отправили невалидный архив (rar вместо zip), не приложили нужные файлы (или наоборот приложили лишние - временные файлы от Mac OS и т.п.), рекомендуется проверить содержимое архива в консоли:

```
unzip -l your_solution.zip
```

Если Вы столкнулись с какой-то из них присылайте ссылку на выполненное задание (Job) в чат курса. Пример ссылки:

<https://everest.distcomp.org/jobs/67893456230000abc0123def>

Как правильно настроить окружение, чтобы оно совпадало с тестовым окружением?

1. Если еще не установлено, то установите conda  
<https://docs.conda.io/projects/conda/en/latest/user-guide/install/>
2. Настройте окружение для разработки на основе README.md курса  
<https://github.com/big-data-team/python-course>
3. Скачайте необходимые датасеты для выполнения задания  
<https://github.com/big-data-team/python-course#study-datasets>



## 6. Дополнительные задания (не на оценку)

Обратите внимание на возможности использования `call` из модуля `unittest.mock`. С его помощью можно подготовить список всех обращений к интересующему методу или объекту класса (даже с учетом вложенных вызовов методов) и убедиться, что ровно этот набор методов с интересующими аргументами и был вызван.

Задание:

1. Создайте mock на вызов функции `load_asset_from_file`;
2. Посмотрите на все вызовы к полученному объекту, если вызвать `process_cli_arguments` с аргументом `periods=[1, 2, 5]`;
3. Создайте нужные вызовы `call` и убедитесь, что ровно они и были вызваны в указанном порядке.