

# Шпаргалка: оценка покрытия

Оценка покрытия — это метрика. Она показывает, насколько полно код покрыт тестами.

**Пример.** Сервис вычисляет зарплату менеджера по продажам. Сотрудник получает 5% от всех продаж за месяц. Общая сумма не может превышать пятьдесят тысяч:

```
class SalaryService:

    @staticmethod
    def calculate_salary(sales):
        percent = 5
        salary = sales * percent / 100
        salary_limit = 50000
        if salary > salary_limit:
            salary = salary_limit
        return salary
```

Тестировщик написал юнит-тест для этого сервиса:

```
from salary import SalaryService

class TestSalaryService:
    def test_calculate_salary_when_under_limit(self):
        actual = SalaryService.calculate_salary(50000)
        expected = 2500
        assert actual == expected
```

Нужно понять, насколько хорошо метод `calculate_salary()` покрыт тестами.



**Покрывание кода** говорит, какой процент программы выполняется во время тестов.

Можно смотреть на процент покрытия:

- строк кода,

- условных операторов,
- методов.

## Процент покрытых строк кода

Строка кода считается покрытой, если она выполнялась во время теста хотя бы раз.

**Пример:** строка `salary = sales * percent / 100` покрыта. Она выполняется каждый раз, когда ты вызываешь метод `calculate_salary()`.

Строка `salary = salary_limit` не покрыта — она выполняется не всегда. Если зарплата не превышает лимит, код внутри `if` не выполняется.

## Процент покрытых условий

Условие считается покрытым, если каждая ветвь решения выполнялась.

**Пример:** условие `if salary > salary_limit` покрыто частично: тест проверяет только вариант с зарплатой ниже лимита.

## Процент покрытых методов

Метод считается покрытым, если он выполнялся при тестировании хотя бы один раз.

**Пример:** метод `calculate_salary()` вызывается в тесте. Значит, он покрыт.

## Какой критерий покрытия считается хорошим



Ни один из способов не даёт полной картины реального покрытия.

Чтобы убедиться, что модуль протестирован на 100%, нужно комбинировать разные методы. Юнит-тесты дополняют другими видами тестов — функциональными, интеграционными, приёмочными.

## От чего зависит покрытие

Покрытие зависит от состояния проекта и подходов, которые использует команда.

На молодых проектах сразу начинают покрывать код тестами. Поэтому процент покрытия высокий — 80-90% и больше.

На длительных проектах даже 30% — неплохой результат. Обычно здесь много легаси-кода. Его сложно поддерживать и тестировать, потому что никто в команде не знает, как он работает. Исходный код покрывают тестами при случае — на рефакторинге или при доработке. В основном тестируют новые функции и методы.

## К какому проценту стремиться

Высокий процент покрытия — это хорошо. Но он не должен превращаться в самоцель. Кроме количественного покрытия, должно быть ещё и качественное. Поэтому важно использовать техники тест-дизайна, когда пишешь тесты и подбираешь тестовые данные.

В реальных проектах сложно достичь покрытия 100%. Обычно выбирают другой минимально необходимый процент покрытия — например, 80%.

## Как считать покрытие

Автоматизировать оценку покрытия помогает **плагин** `pytest-cov`. Ты вводишь команду, а программа создаёт отчёт: показывает, какие части кода протестированы, а какие нет.

**Плагин** — это модуль, который расширяет возможности программы. Его нужно подключать отдельно.

Чтобы запросить анализ, введи в терминал PyCharm команду:

```
pytest --cov=<название_файла>
```

**Пример:** команда `pytest --cov=salary` проанализирует покрытие для файла `salary.py`.

Вывод выглядит примерно так:

Name	Stmts	Miss	Cover
-----			
salary.py	9	1	89%

```
-----  
TOTAL          9      1    89%
```

Здесь:

- **Stmts** — сколько строк кода содержит файл.
- **Miss** — сколько строк не участвует в тестах.
- **Cover** — общий процент покрытия.

Отчёт говорит, что в программе 9 строк. Одна из них не участвует в тесте. Тест покрывает 89% кода.

## Параметры запуска

Кусочек `=salary` в команде — это параметр. Он говорит плагину, где искать файлы для анализа.

В параметре можно указать название директории. Например, `pytest --cov=dir`. Плагин проверит покрытие всех файлов, которые там хранятся. Такую команду лучше запускать из родительского каталога: так система точно найдёт нужную папку.

Параметр указывать не обязательно. Если запустить команду без него, плагин проанализирует файлы текущей директории:

```
pytest --cov
```

Если добавить к команде опцию `--cov-branch`, анализ будет учитывать ветвление кода.

### Пример:

```
# оценить покрытие тестами файла salary.py с учётом ветвления  
$ pytest --cov=salary --cov-branch
```

Получится такой результат:

Name	Stmts	Miss	Branch	BrPart	Cover
-----	-----	-----	-----	-----	-----
salary.py	9	1	2	1	82%
-----	-----	-----	-----	-----	-----
TOTAL	9	1	2	1	82%

Добавились две колонки:

- **Branch** — сколько строк кода в условиях.
- **BrPart** — сколько из них не покрыто.

## Как сохранить отчёт

Чтобы сохранить отчёт, добавь к команде опцию `--cov-report`.

**Пример:**

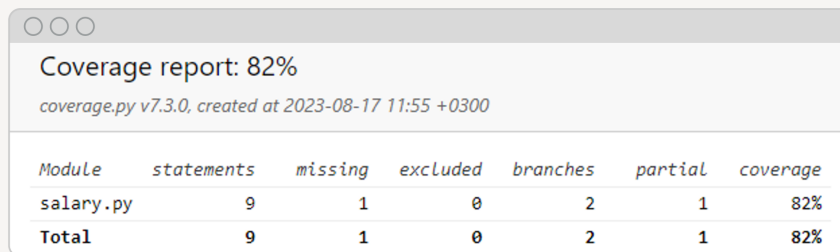
```
# оценить покрытие файла salary.py с учётом ветвления и сохранить отчёт в html
pytest --cov=salary --cov-branch --cov-report=html
```

Параметр `=html` уточняет, что отчёт нужен в формате `html`. Вместо него можно выбрать `xml`, `json` и `lcov`.

Когда запустишь команду, плагин автоматически создаст папку для отчётов. Она появится в директории, откуда прошёл вызов команды.

Когда отчёт будет готов, терминал выведет сообщение `Coverage HTML written to dir` `htmlcov`. Это значит: отчёт о покрытии сохранён в формате `html` в директории `htmlcov`.

**Пример:** плагин сохранит отчёт для `salary.py` в файле `index.html`. Он выглядит так:



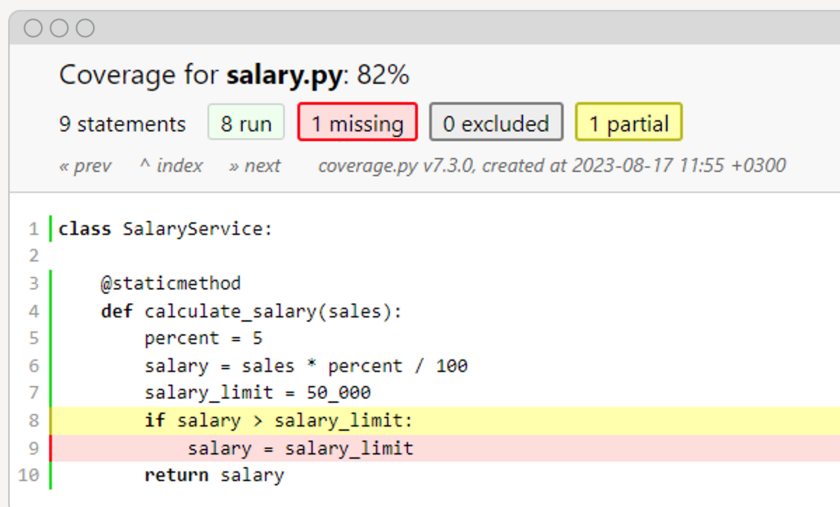
Coverage report: 82%

*coverage.py v7.3.0, created at 2023-08-17 11:55 +0300*

Module	statements	missing	excluded	branches	partial	coverage
salary.py	9	1	0	2	1	82%
<b>Total</b>	<b>9</b>	<b>1</b>	<b>0</b>	<b>2</b>	<b>1</b>	<b>82%</b>

Чтобы получить больше информации, кликни по названию файла. Ты увидишь код целиком.

**Пример:**



Coverage for **salary.py**: 82%

9 statements   8 run   1 missing   0 excluded   1 partial

« prev   ^ index   » next   *coverage.py v7.3.0, created at 2023-08-17 11:55 +0300*

```

1 | class SalaryService:
2 |
3 |     @staticmethod
4 |     def calculate_salary(sales):
5 |         percent = 5
6 |         salary = sales * percent / 100
7 |         salary_limit = 50_000
8 |         if salary > salary_limit:
9 |             salary = salary_limit
10 |         return salary

```

Цвета строк означают полноту покрытия. Зелёный — покрыто тестами, жёлтый — покрыто частично, красный — не покрыто совсем.

Больше о возможностях плагина — [в документации](#).