Рефакторинг текста:

```
from typing import List, Tuple
from statistics import mean

# Определение классов данных
class СинтаксическаяКонструкция:
    def __init__(self, id: int, название: str, описание: str, язык_id: int):
        self.id = id
        self.название = название
        self.описание = описание
        self.язык_id = язык_id

class ЯзыкПрограммирования:
    def __init__(self, id: int, название: str):
        self.id = id
        self.название = название

class КонструкцииЯзыка:
    def __init__(self, конструкция_id: int, язык_id: int):
        self.конструкция_id = конструкция_id
        self.язык_id = язык_id

# Функция для получения конструкций, заканчивающихся на 'ие'
def get_constructions_with_suffix(constructions: List[СинтаксическаяКонструкция],
                                  languages: List[ЯзыкПрограммирования],
                                  suffix: str) -> List[Tuple[str, str]]:
    return [
        (конструкция.название, язык.название)
        for конструкция in constructions
        for язык in languages
        if конструкция.язык_id == язык.id and конструкция.название.endswith(suffix)
    ]

# Функция для расчета средней длины описания конструкций по языкам
def get_average_description_length_by_language(constructions: List[СинтаксическаяКонструкция],
                                               languages: List[ЯзыкПрограммирования]) -> List[Tuple[str, float]]:
    result = [
        (язык.название, mean([len(конструкция.описание) for конструкция in constructions if
конструкция.язык_id == язык.id]))
        for язык in languages
    ]
    return sorted(result, key=lambda x: x[1])
```

```
# Функция для получения языков, начинающихся на определённый префикс, и их конструкций
def get_languages_and_constructions_by_prefix(constructions: List[СинтаксическаяКонструкция],
                                              languages: List[ЯзыкПрограммирования],
                                              prefix: str) -> List[Tuple[str, List[str]]]:
    return [
        (язык.название, [конструкция.название for конструкция in constructions if конструкция.язык_id
== язык.id])
        for язык in languages
        if язык.название.startswith(prefix)
    ]

# Создание тестовых данных
языки = [
    ЯзыкПрограммирования(1, 'Python'),
    ЯзыкПрограммирования(2, 'JavaScript'),
    ЯзыкПрограммирования(3, 'C++')
]

конструкции = [
    СинтаксическаяКонструкция(1, 'Цикл', 'Повторение операций', 1),
    СинтаксическаяКонструкция(2, 'Условие', 'Выбор выполнения', 1),
    СинтаксическаяКонструкция(3, 'Функция', 'Определение подпрограммы', 2),
    СинтаксическаяКонструкция(4, 'Массив', 'Коллекция данных', 3)
]

# Выполнение функций
result_1 = get_constructions_with_suffix(конструкции, языки, 'ие')
print("Список конструкций, заканчивающихся на 'ие':", result_1)

средняя_длина_описания = get_average_description_length_by_language(конструкции, языки)
print("Средняя длина описания конструкций по языкам:", средняя_длина_описания)

result_3 = get_languages_and_constructions_by_prefix(конструкции, языки, 'P')
print("Список языков, начинающихся на 'P', и их конструкции:", result_3)
```

—----------

Модульные тесты:

```
import unittest
```

```
class TestLanguageConstructions(unittest.TestCase):
    def setUp(self):
        self.languages = [
            ЯзыкПрограммирования(1, 'Python'),
            ЯзыкПрограммирования(2, 'JavaScript'),
            ЯзыкПрограммирования(3, 'C++')
        ]
        self.constructions = [
            СинтаксическаяКонструкция(1, 'Цикл', 'Повторение операций', 1),
            СинтаксическаяКонструкция(2, 'Условие', 'Выбор выполнения', 1),
            СинтаксическаяКонструкция(3, 'Функция', 'Определение подпрограммы', 2),
            СинтаксическаяКонструкция(4, 'Массив', 'Коллекция данных', 3)
        ]

    def test_get_constructions_with_suffix(self):
        result = get_constructions_with_suffix(self.constructions, self.languages, 'ие')
        expected = [('Условие', 'Python')]
        self.assertEqual(result, expected)

    def test_get_average_description_length_by_language(self):
        result = get_average_description_length_by_language(self.constructions, self.languages)
        expected = [('C++', 17), ('JavaScript', 22), ('Python', 23)]
        self.assertEqual(result, expected)

    def test_get_languages_and_constructions_by_prefix(self):
        result = get_languages_and_constructions_by_prefix(self.constructions, self.languages, 'P')
        expected = [('Python', ['Цикл', 'Условие'])]
        self.assertEqual(result, expected)

if __name__ == '__main__':
    unittest.main()
```