УДК 004.422.833

**СОВМЕСТНОЕ ИСПОЛЬЗОВАНИЕ С++ И PYTHON ДЛЯ РАЗРАБОТКИ ЭФФЕКТИВНЫХ ПРОГРАММ**

Грищенко Д.А., Данильчук К. М., Моргунов А. Г.

ГОУ ВПО «Донецкий национальный технический университет»

(г. Донецк)

[Darya.grischenko@gmail.com](mailto:Darya.grischenko@gmail.com), [kirill.danilchuk01@mail.ru](mailto:kirill.danilchuk01@mail.ru) [mag17122000@mail.ru](mailto:mag17122000@mail.ru)

***Грищенко Д. А., Данильчук К. М., Моргунов А. Г.******Совместное использование C++ и Python для разработки эффективных программ.*** *Статья посвящена совместному использованию языков программирования C++ и Python, что позволяет находить компромисс в скорости разработки и скорости выполнения. В статье выполнено сравнение скорости работы эквивалентных програмах на C++ и Python.*

***Ключевые слова:*** *C++, Python, Производительность, Скорость разработки, CMake, PyBind11.*

**Введение**

В связи с активным развитием компьютерных технологий и их широким внедрением в повседневную жизнь, растут и требования к скорости разработки программного обеспечения и его быстродействию. Для того чтобы обеспечить запросы реальных пользователей и поддерживать конкурентоспособность, необходимо найти баланс между этими двумя важными критериями. В данной статье рассматривается один из вариантов решения проблемы, а именно совместное использование таких языков программирования как С++ и Python.

**Объединение С++ и Python зачем это нужно и как это используется на практике.**

С++ является языком, который реализует подход называемый “Zero overhead” – не нужно платить за то, что не используется [1]. За счет этого производительность программ является высокой, но скорость разработки из-за этого страдает, т.к. всё требуется настраивать вручную.

Язык Python [2] является абсолютной противоположностью C++, на котором приложения пишутся быстро, однако работают медленно. Цель статьи – продемонстрировать методы разработки программного обеспечения объединяющие преимущества обоих языков.

При написании высоконагруженных приложений необходимо быстрое исполнение программного кода без дополнительных расходов на сборку мусора, интерпретацию кода и прочего. В большинстве случаев такие приложения разрабатываются на C++, но часто бывают подзадачи, не требующие высокой скорости работы или просто не являющиеся вычислительно сложными, например, узнать время из интернета или просмотреть погоду. Такие задачи можно решить на С++, но более правильно, с точки зрения оптимальности времени разработки, их решать, например, на таком языке как Python, используя уже готовые решения из существующих библиотек или реализуя собственные идеи.

При разработке программ на Python иногда встречаются места, где требуется высокая производительность. Это можно обеспечить за счет реализации функции на C++ и вызова ее из Python.

Существует несколько способов совместного использования языков C++ и Python при разработке приложений. Для этого можно использовать следующие библиотеки:

* C library python;
* Boost.Python;
* PyBind.

C library python – это низкоуровневая API, которая предоставляет доступ к Python на уровне библиотеки реализации [3].

Boost.Python – требует Boost (слишком большая зависимость), является обёрткой над C library python [4].

PyBind – аналог Boost.Python, предоставляет все тоже самое, но не требует зависимости от Boost (объект исследования данной статьи) [5].

Рассмотрим методы и примеры реализации данного подхода.

**Как вызывать Python код из C++ и почему это хорошо (network time).**

Для вызова программного кода на языке Python внутри приложения написанного на C++ с получением результата вызова, необходим поток выполнения программы изображенный на рисунке 1.

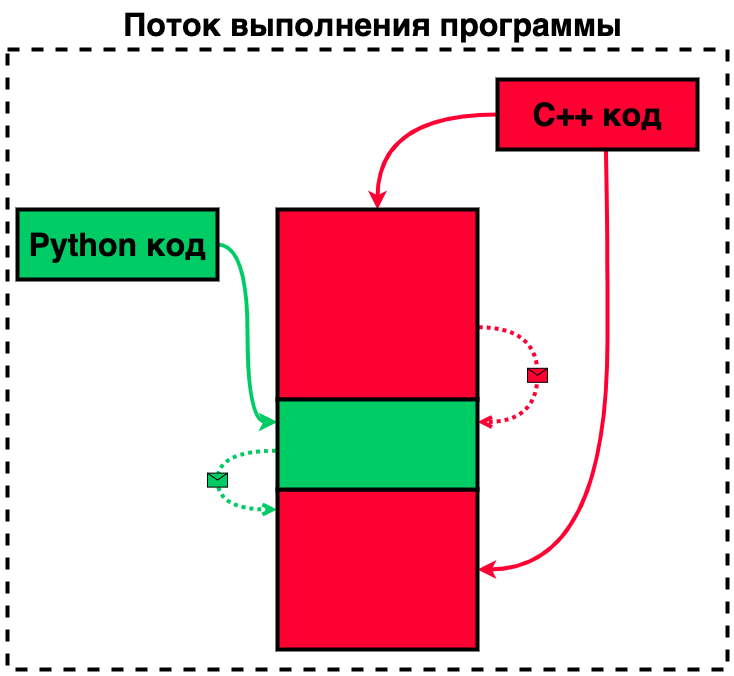


Рисунок 1 – Вызов Python из C++

Рассмотрим реализацию такого подхода на примере задачи, в которой необходимо получить время из интернета, например, чтобы проверить не закончился ли пробный период пользователя (trial version), для приложений, которые предоставляют бесплатную пробную версию.

Структура проекта для реализации описанной задачи приведена на рисунке 2.

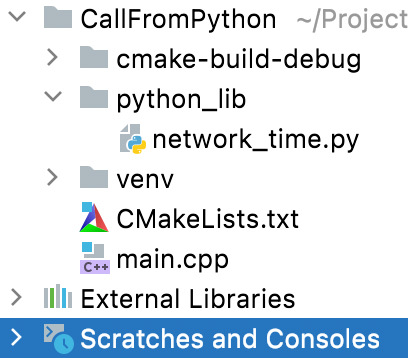


Рисунок 2 – Структура проекта

Реализация функции приведена на рисунке 3.

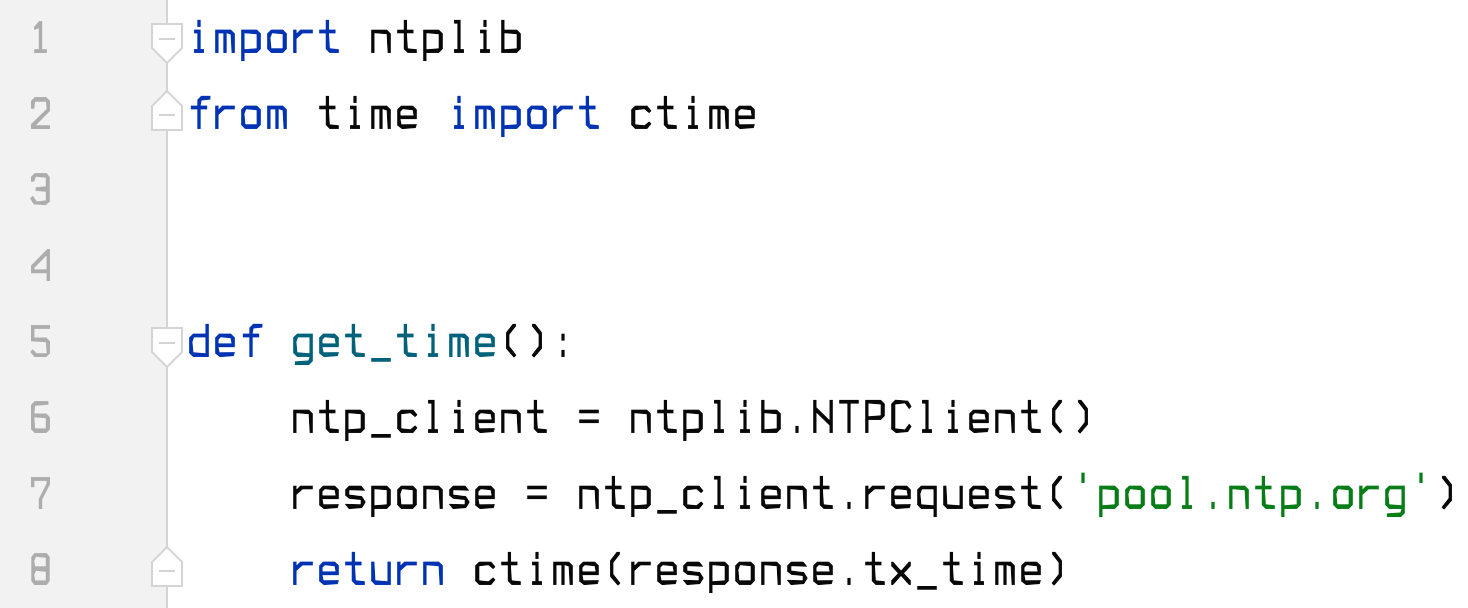


Рисунок 3 – Пример функции на Python

Проекты на С++ можно создавать разным способом (зависит от системы сборки). Самая популярная система сборки – CMake. Создадим CMake-файл как на рисунке 4.

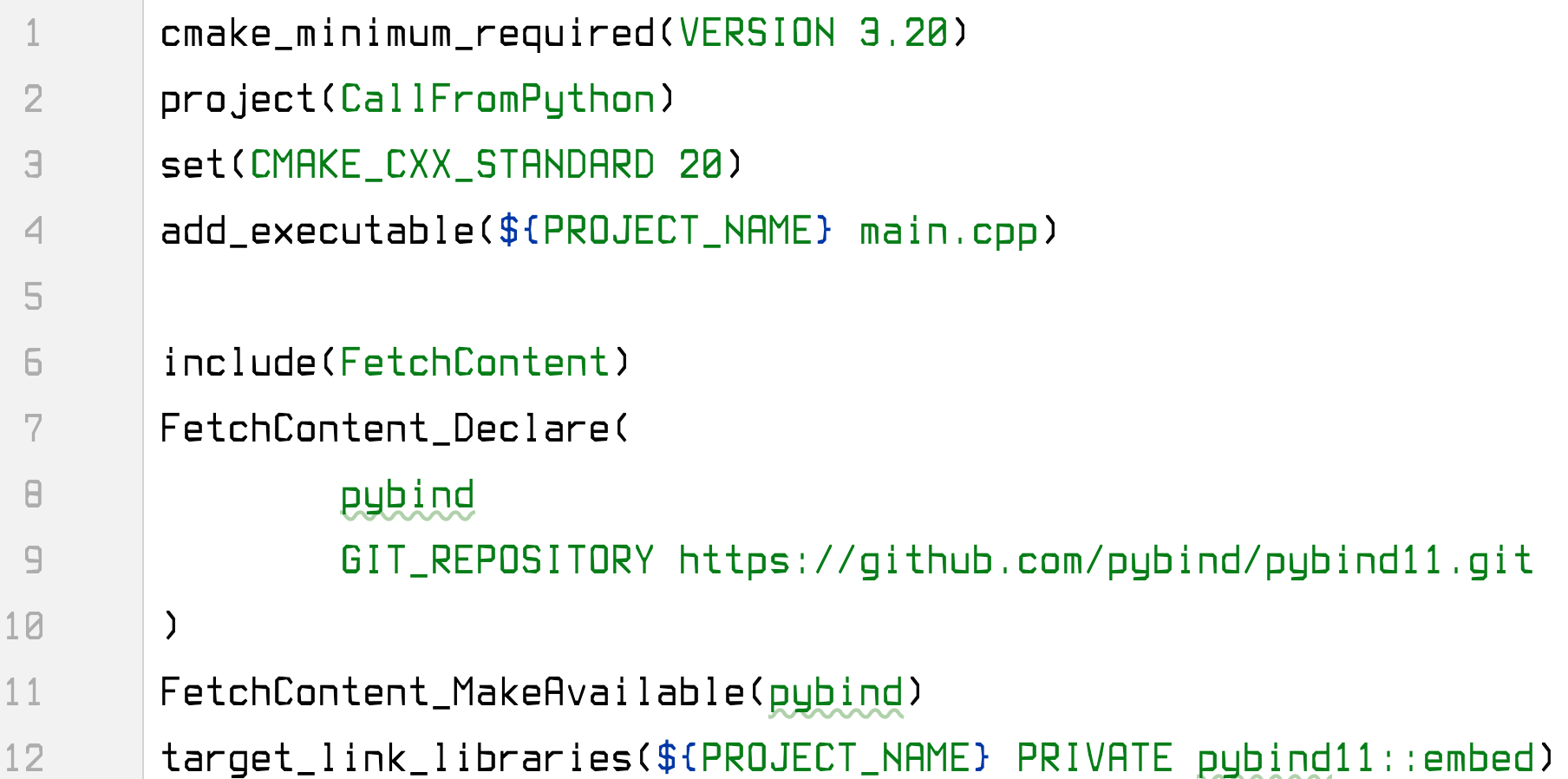


Рисунок 4 – CMake файл C++ проекта

Первые 4 строки создают стандартный C++ проект. Строка 6 включает библиотеку CMake для получения библиотек из Git репозиториев. Строки 7-10 скачивают библиотеку. Строка 11 делает библиотеку доступной для использования. Строка 12 подключает библиотеку к проекту.

Создадим main.cpp с исходным кодом на C++ (см. рис. 5).

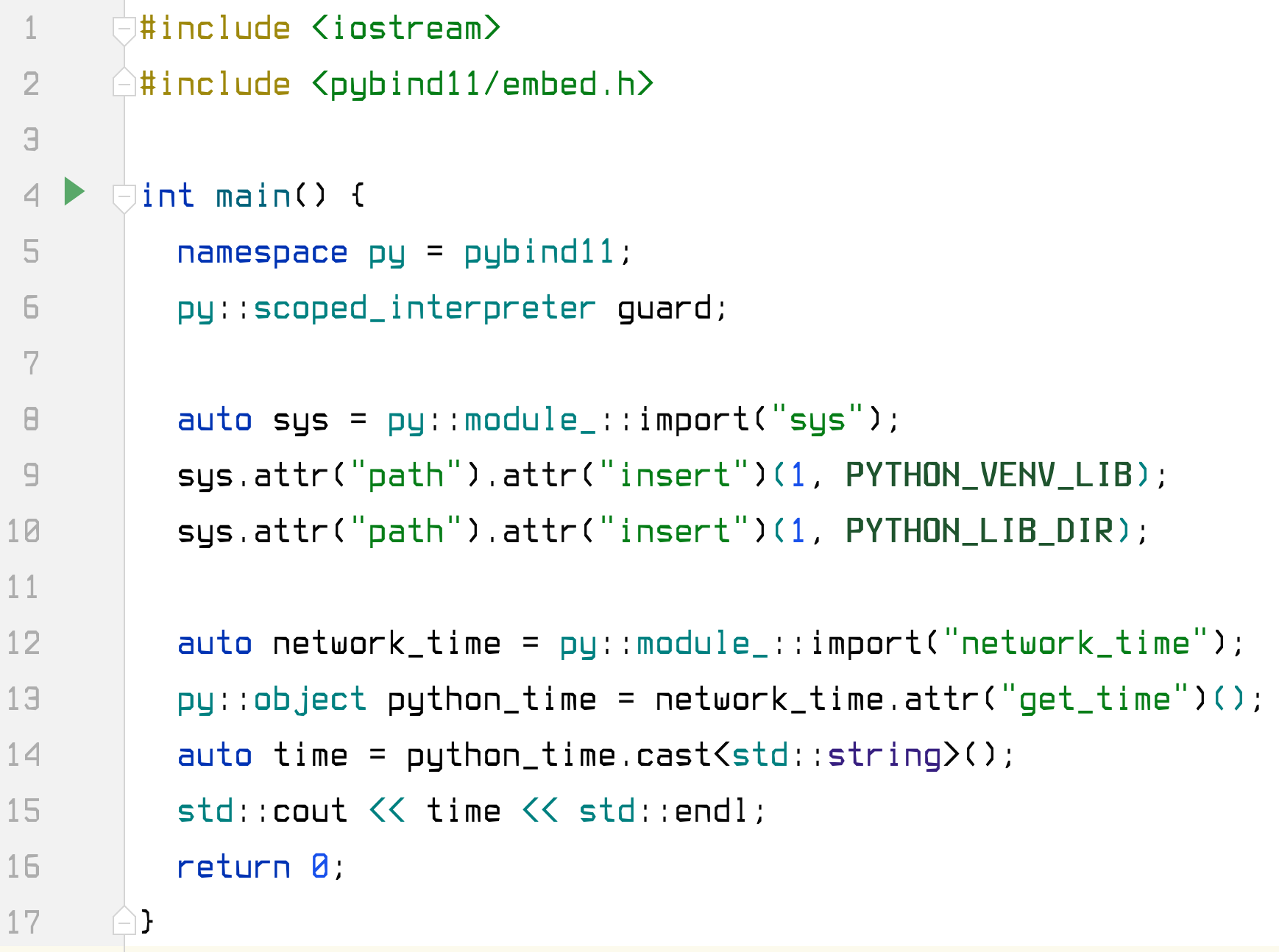


Рисунок 5 – Код проекта

Приведенный выше код запускает интерпретатор Python, настраивает его и далее вызывает функцию get\_time из модуля network\_time (см. рис. 3). Полученный результат конвертируется в std::string в переменную time и передается на стандартный поток вывода. Разделение вызовов C++ и Python приведено на рисунке 6.



Рисунок 6 – Разделение кода

Строки 1 и 2 подключают необходимые заголовочные файлы. Строка 6 создаёт интерпретатор Python, который разрушится при выходе из функции main (принцип RAII [6]). Строка 8 загружает стандартный модуль Python – sys. Для получения атрибутов из Python существует функция attr. Строки 9 и 10 настраивают пути к виртуальному окружению и к пользовательским файлам Python. Строка 12 загружает пользовательский модуль network\_time. Строка 13 вызывает функцию get\_time из модуля network\_time, результатом которого является py::object – структура данных, которая хранит переменные в Python. Строка 14 – конвертирует python\_time в std::string. Строка 15 вывод данные на консоль (см. рис. 7).

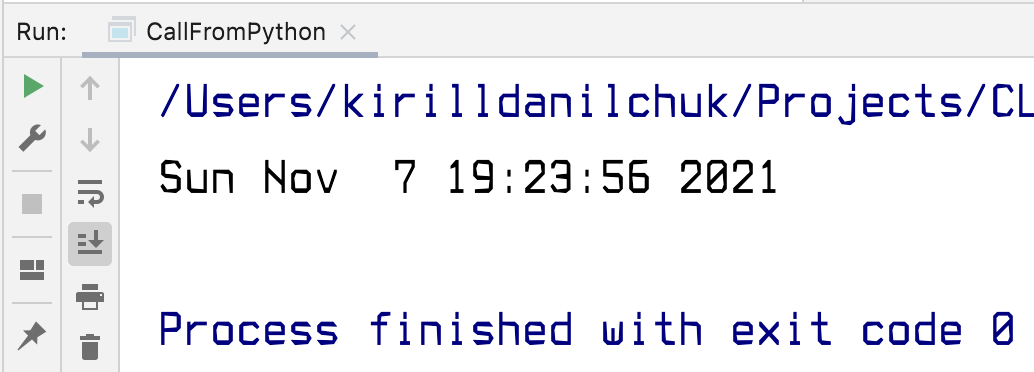


Рисунок 7– Результат выполнения программы

В итоге удалось получить время из интернета максимально простыми средствами, благодаря библиотеке на Python, но при этом основным языком приложения остался C++.

**Как вызывать C++ код из Python и почему это хорошо.**

Когда все приложение пишется на Python, но возникает место в программе, где нужна максимальная скорость работы, например необходимо посчитать большое количество тригонометрических формул для рисования окружностей, то выполнение подобных задач лучше всего делегировать функциям, написанных на C++ (см. рис. 8).

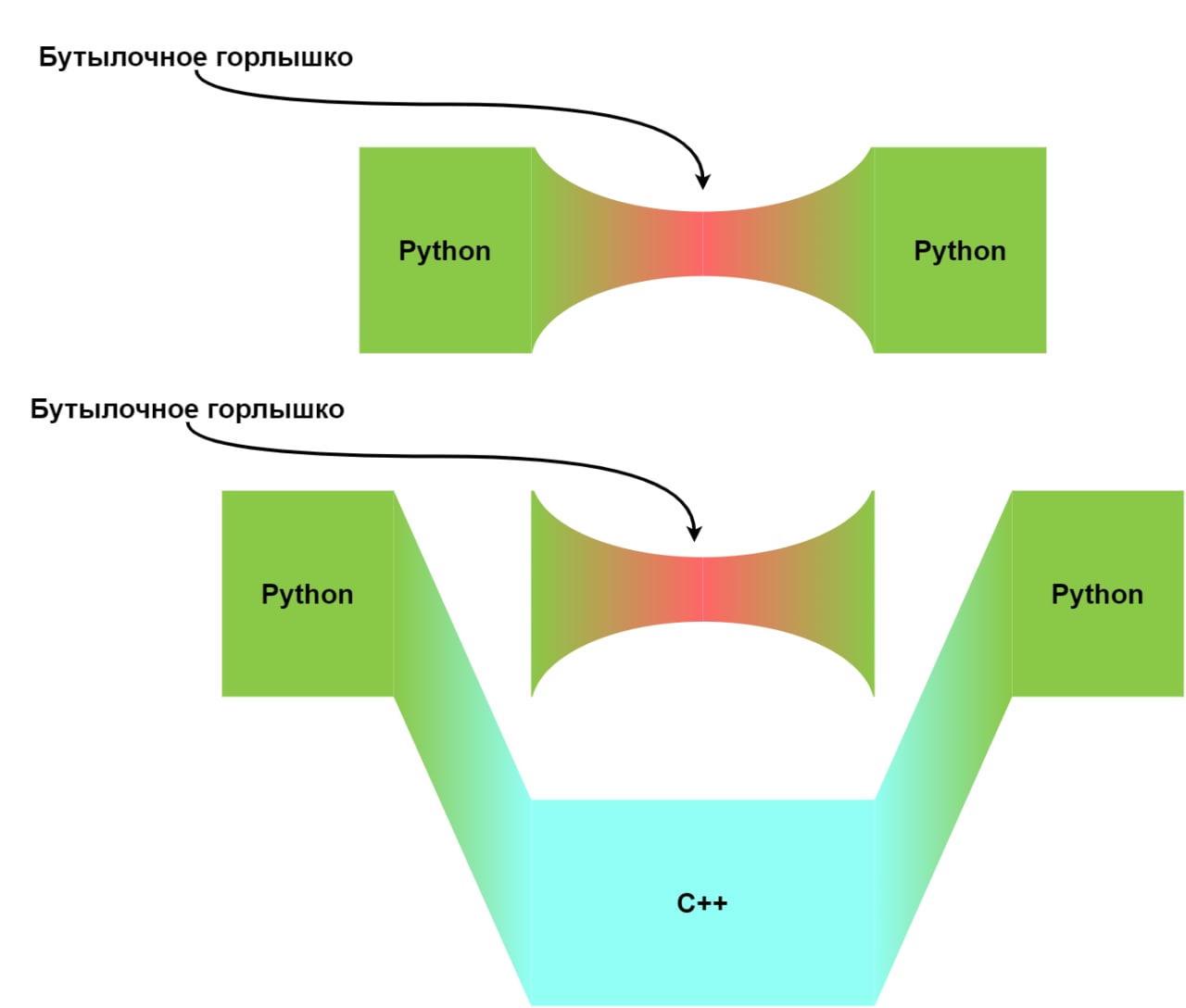


Рисунок 8 – Проблема производительности

Если возникает необходимость оптимизации «бутылочного горлышка», то создают динамические библиотеки на более производительном языке. В данной статье рассмотрим решение задачи на языке С++.

Предположим, в приложении возникла необходимости оптимизировать вычисление суммы элементов массива. Создадим проект С++ со следующей структурой (см. рис. 9).

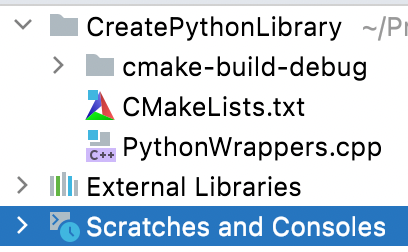


Рисунок 9 – Структура проекта

CMake файл минимально отличается от предыдущего (см. рис. 10).

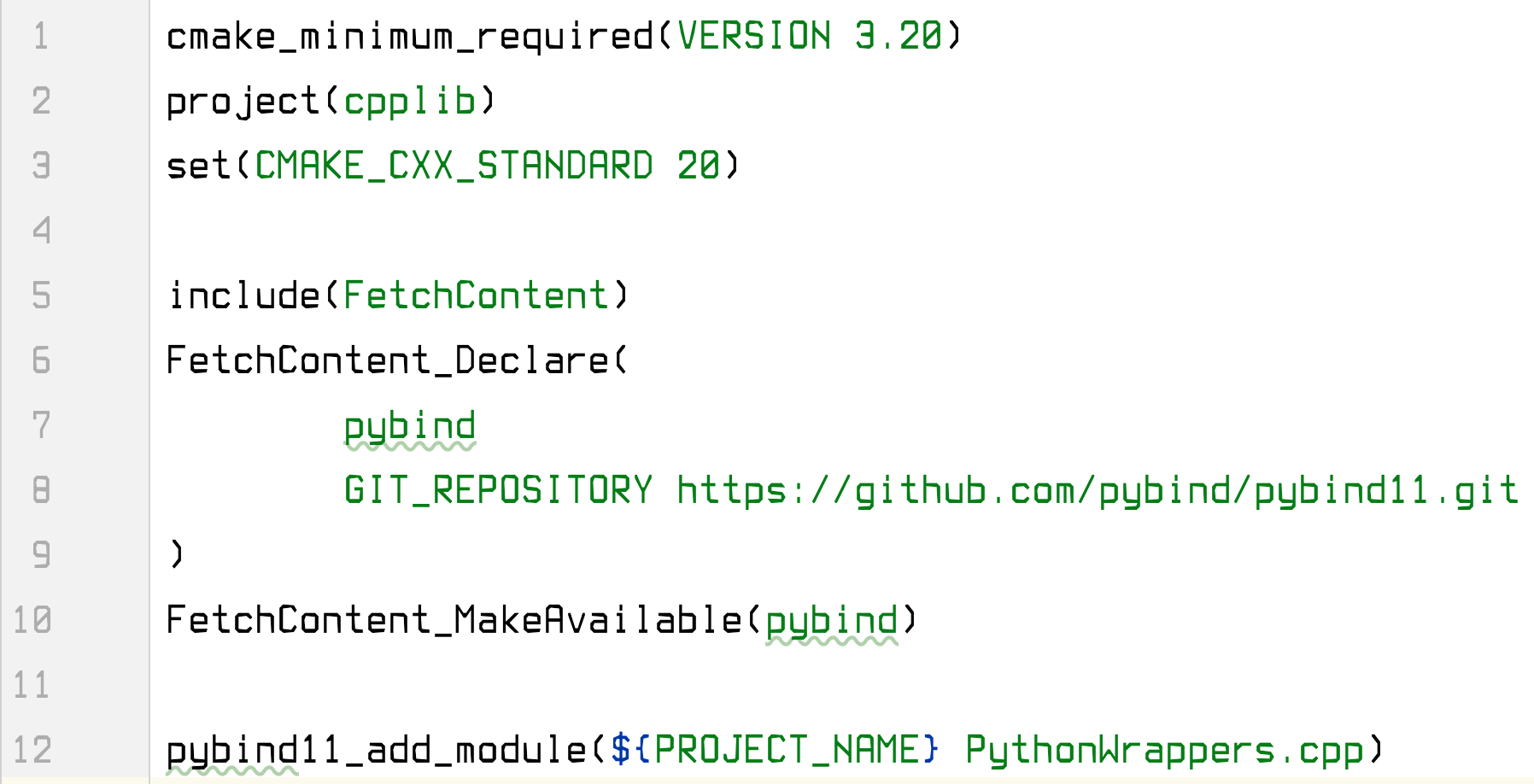


Рисунок 10 – CMake файл

Как можно видеть из рисунка выше, добавилась 12 строка, в которой вызывается функция из проекта PyBind, в которую добавляются cpp-файлы для создания динамической библиотеки для Python. Код библиотеки представлен на рисунке 11.

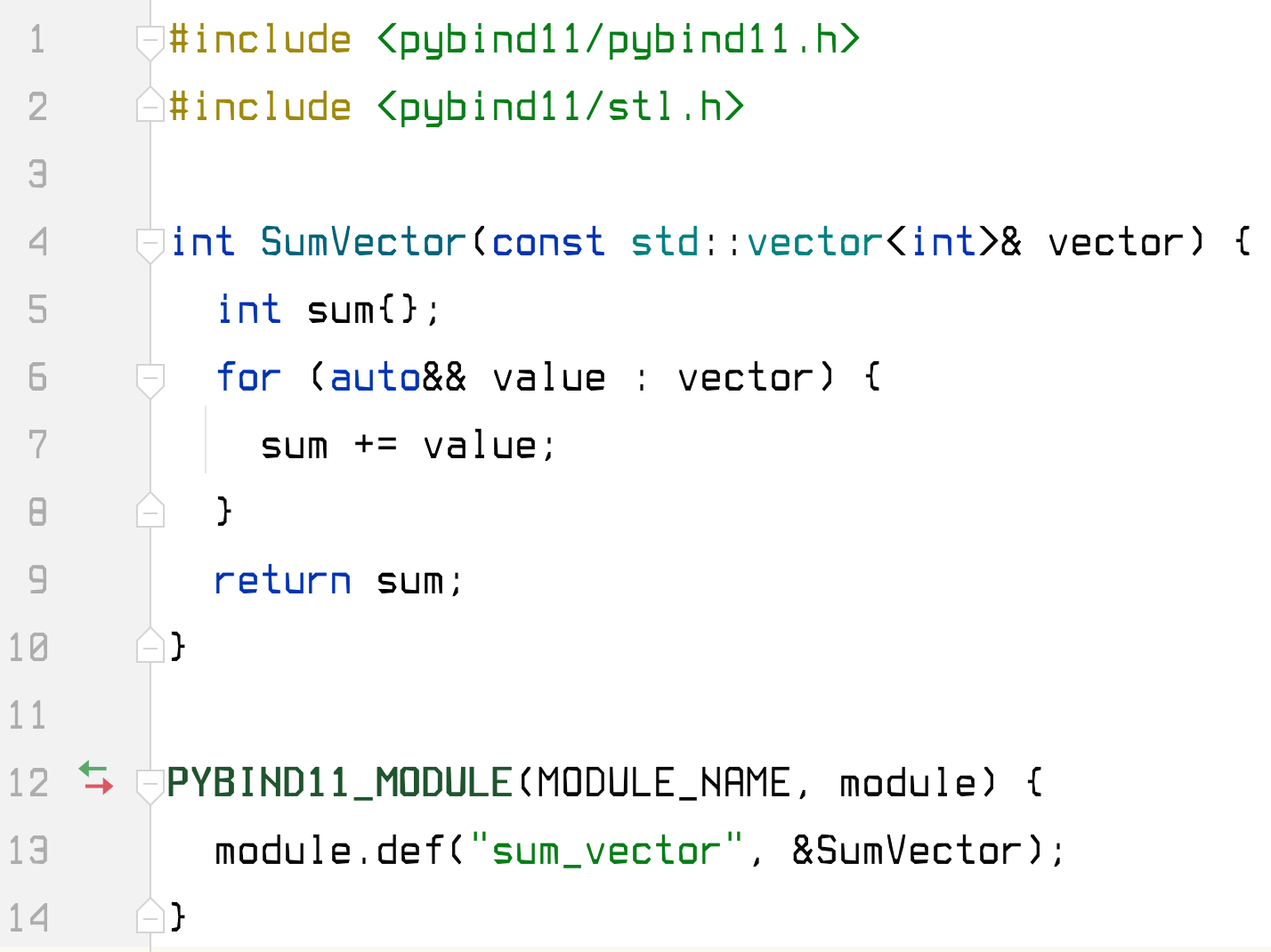


Рисунок 11 – Код библиотеки

Строка 1 и 2 подключают заголовочные файлы библиотеки. Строки с 4 по 10 просто описывают функцию, которая возвращает сумму элементов вектора (простая реализация, как пример). Строка 12 определяет макрос, параметрами которого является название модуля для Python, второй параметр – название обработчика, который позволяет получить API библиотеки. Строка 13 определяет функцию с названием sum\_vector в модуле, которая будет вызывать SumVector из библиотеки C++.

После компиляции проекта, появится файл с библиотекой (название и расширения файла зависит от архитектуры, например на MacBook с процессором M1 файл имеет название cpplib.cpython-39-darwin.so). Данный файл необходимо скопировать в проект с Python, где находятся все библиотеки в виртуальном окружении это папка site-pakages. После копирования можно пользоваться библиотекой (см. рис. 11).

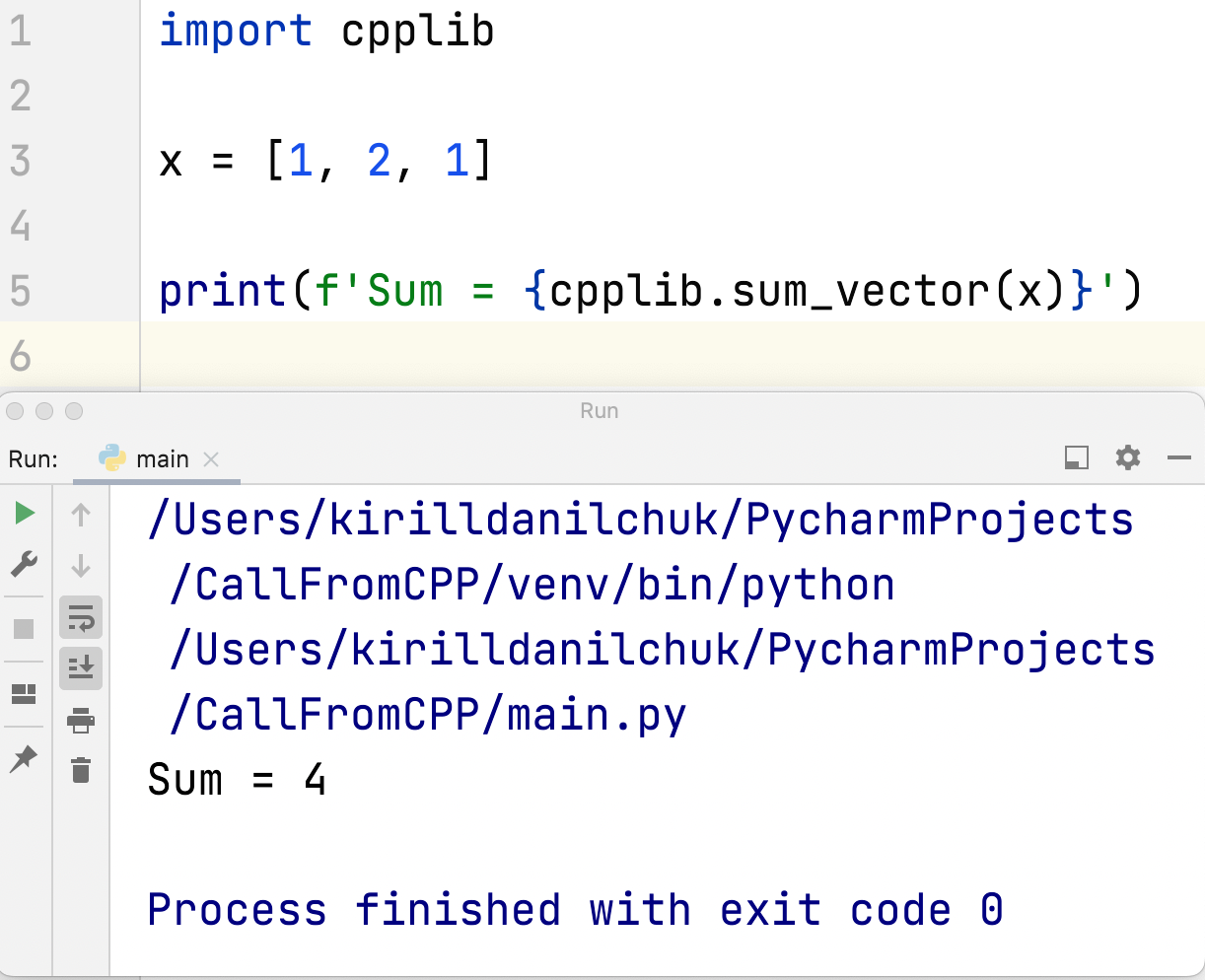


Рисунок 11 – Пример использования библиотеки

Существует проблема с IDE, если после выполнения данных шагов в IDE не появляются подсказки (в PyCharm все работает по умолчанию), то для этого необходимо создать интерфейсный файл и поместить его рядом с файлом библиотеки. Данные файлы имеют расширение .pyi (python interface) и называются stub. Данные файлы можно создавать вручную, а можно и сгенерировать из файла библиотеки. Пример файла для C++ библиотеки приведен на рисунке 12.

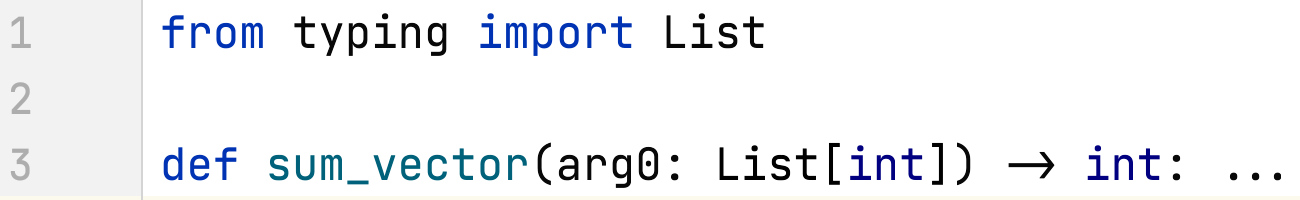


Рисунок 12 – Stub-файл

Для генерации данного файла нужно установить библиотеку MyPy и вызвать команду из папки, где установлена библиотека С++,   
как на рисунке 13.



Рисунок 13 – Команда для генерации stub-файла

В итоге удалось написать Python приложение с более быстрым подсчетом суммы, за счет вызова уже скомпилированного C++ кода в виде динамической библиотеки.

**Сравнение производительности C++ и Python.**

Для сравнения производительности C++ и Python будем считать суммы тригонометрических функций последовательно и параллельно.

На Python для параллельных вычислений (в разных потоках) использовался ThreadPool и Executor, а на C++ потоки. Для того, чтобы тестирование было более корректным, функции на C++ обернем в Python и всё тестирование будем производить внутри Python. Исходных код тестирования и результаты тестирования доступны на GitHub [7].

В процессе тестирования были получены следующие результаты: время выполнения на Python в последовательном режиме – 6.51 секунд, в параллельном 6.35 секунд; время выполнения на C++ в последовательном режиме 0.39 секунды, в параллельном - 0.13 секунды. Исходя из полученных результатов можно судить, что C++ в последовательном режиме в 16 раз быстрее, а в параллельном в 48.

**Выводы**

В статье рассмотрены методы создания приложений на языке Python с использованием С++ вставок и наоборот на базе библиотеки PyBind11, обладающей всем необходимым функционалом для создания подобных проектов.

Было выполнено сравнение производительности одного и того же алгоритма, написанного на чистом Python и с использованием С++ вставок. Тестирование показало, что комбинированное приложение выполняется в 16 раз быстрее в однопоточном режиме и в 48 раз в многопоточном.

На основе проведенного анализа можно сделать вывод о том, что представленный подход позволяет достичь баланса между высокой скоростью разработки приложений на Python и производительностью С++ программ.

Список литературы

1. Zero-overhead principle [электронный ресурс]. – Режим доступа: <https://en.cppreference.com/w/cpp/language/Zero-overhead_principle> (дата обращения 07.11.2021).
2. Python [электронный ресурс]. – Режим доступа: <https://www.python.org/> (дата обращения 07.11.2021).
3. Extending Python with C or C++ [электронный ресурс]. – Режим доступа: <https://docs.python.org/3/extending/extending.html> (дата обращения 07.11.2021).
4. Boost.Python [электронный ресурс]. – Режим доступа: <https://www.boost.org/doc/libs/1_70_0/libs/python/doc/html/index.html> (дата обращения 07.11.2021).
5. PyBind11 [электронный ресурс]. – Режим доступа: <https://pybind11.readthedocs.io/en/latest/> (дата обращения 07.11.2021).
6. RAII [электронный ресурс]. – Режим доступа: <https://en.cppreference.com/w/cpp/language/raii/> (дата обращения 07.11.2021).
7. Исходный код тестирования [электронный ресурс]. – Режим доступа: <https://github.com/KirillDanilchuk/CompareCppAndPythonSpeed> (дата обращения 20.11.2021).

***Грищенко Д. А., Данильчук К. М., Моргунов А. Г.******Совместное использование C++ и Python для разработки эффективных программ.*** *Статья посвящена совместному использованию языков программирования C++ и Python, что позволяет находить компромисс в скорости разработки и скорости выполнения. В частности, в статье выполнено сравнение скорости работы эквивалентных програмах на C++ и Python.*

***Ключевые слова:*** *C++, Python, Производительность, Скорость разработки, CMake, PyBind11.*

***Grishchenko D. A., Danilchuk K. M., Morgunov A. G.******Joint use of C ++ and Python for the development of effective programs.*** *This article is devoted to the joint use of the C ++ and Python programming languages, which allows you to find a compromise in development speed and execution speed. In particular, the article compares the performance of equivalent programs in C ++ and Python.*

***Keywords:*** *C ++, Python, Performance, Development Speed, CMake, PyBind11.*