

12+

И.В. Боев

Создаём ИИ-агента

Практикум на Python для начинающих



Игорь Боев

Создаем ИИ агента. Практикум

на Python для начинающих

<https://litres.ru/74591984>

ISBN 9785007147484

Аннотация

Книга для тех, кто хочет собрать своего первого ИИ-агента, но не знает как. Достаточно базового знания Python и знакомства с чатами ИИ. За семь глав вы соберёте четыре работающие системы. В книге нет сложных теоретических выкладок и формул и. т. п. Это практическое ознакомление с CrewAI и начало создания своих ИИ агентов.

Не нужно разбираться в устройстве двигателя, чтобы водить машину. Изложение идёт от простого к сложному, каждая практическая часть начинается с внятной постановки задачи.

Содержание

О книге	6
Где взять файлы курса	8
Практический курс по CrewAI	9
Оглавление	10
Глава 1. Что такое CrewAI и зачем он нужен	10
Глава 2. Подготовка окружения	11
Глава 3. Основа: Агент, Задача, Команда	12
Глава 4. Процессы: как команда распределяет работу	13
Глава 5. Инструменты, знания и память	14
Глава 6. Flows: точное управление процессом	15
Глава 7. Итоги	16
Скрипты курса	18
Модели курса	19
Глава 1. Что такое CrewAI и зачем он нужен	20
1.1. Идея: команда сотрудников вместо одного агента	20
1.2. Какие задачи решает — примеры из жизни	25
1.3. Карта курса	29
Глава 2. Подготовка окружения	33
2.1. Отдельное окружение под CrewAI	33
2.2. Подключение моделей и первый запуск	38

Создаем ИИ агента Практикум на Python для начинающих

Игорь Боев

© Игорь Боев, 2026

ISBN 978-5-0071-4748-4

Создано в интеллектуальной издательской системе Ridero

О книге

Практикум по CrewAI для тех, кто начинает с нуля. За семь глав вы соберёте четыре работающие системы: команду ревью кода, исследователя биржевого актива, службу поддержки по документам и поток самокоррекции кода.

Всё на реальных данных и с разбором того, где агенты ошибаются и почему. Требуется базовое знание Python и установленная Ollama.

Книга состоит из двух частей — читайте обе

Эта книга — только половина курса. Вторая половина лежит в **пяти файлах с кодом**, и они не приложение к тексту, а полноценная часть обучения.

В файлах с кодом вас ждёт то, чего нет в книге:

— **пояснения к каждому блоку** — что подключается, зачем и как работает, прямо рядом с соответствующими строками;

— **пометки # ПОПРОБУЙ** — конкретные опыты: что изменить и что при этом произойдёт. Не расплывчатое «поэкспериментируйте», а точное указание, куда смотреть и какой результат ожидать;

— **пометки # ПРИДУМАЙ САМ** — задания без готовых ответов, по три-четыре в конце каждого файла. Некоторые ведут к темам следующих глав, некоторые просят проверить утверждения книги на практике;

— **разбор ловушек** — мест, где легко ошибиться и где ошибались мы сами, пока готовили эти примеры.

Как работать с курсом. Прочитали параграф — открыли соответствующий файл, разобрали комментарии, запустили, поставили опыты из пометок # ПОПРОБУЙ. Только чтение книги без работы с кодом даст поверхностное знакомство вместо умения.

Файлы курса:

Файл	Глава	Что делает
cai_hello_crew.py	2	Первая команда: один агент, одна задача
cai_code_review_crew.py	3	Ревью кода: автор — критик — редактор
cai_stock_research_crew.py	4	Исследование актива в двух режимах процесса
cai_support_crew.py	5	Служба поддержки по инструкции прибора
cai_code_flow.py	6	Поток самокоррекции кода с переключением модели

Код везде рабочий: его можно запускать, ломать и собирать заново. Именно так и стоит поступать — понимание приходит от опытов, а не от чтения.

Где взять файлы курса

Архив со всеми файлами доступен по любой из двух ссылок,

которые вы найдете в конце книги, приложение А «Шпаргалка».

Ссылки ведут на один и тот же архив, берите ту, что удобнее.

Что делать после скачивания. Распакуйте архив — внутри лежат папки с файлами глав. **Скопируйте все папки целиком в корень вашего учебного проекта CrewAI в PyCharm** — того самого, который вы создадите в главе 2. Структуру папок не меняйте: пути в скриптах рассчитаны именно на неё.

Практический курс по CrewAI

Оглавление

Глава 1. Что такое CrewAI и зачем он нужен

1.1. Идея: команда сотрудников вместо одного агента — Кто и когда сделал CrewAI — 1.1.1. Чем CrewAI отличается от ручной сборки на LangGraph — 1.1.2. Метафора «отдел с ролями» — 1.1.3. Когда CrewAI удобнее, а когда лучше LangGraph

1.2. Какие задачи решает — примеры из жизни — 1.2.1. Команда ревью кода: автор, критик, редактор — 1.2.2. Исследование темы и подготовка отчёта — 1.2.3. Многошаговые процессы с несколькими исполнителями

1.3. Карта курса — 1.3.1. Что освоим по главам — 1.3.2. Возможности CrewAI одним взглядом

Глава 2. Подготовка окружения

2.1. Отдельное окружение под CrewAI — 2.1.1. Почему CrewAI нужна своя кухня — 2.1.2. Создаём проект и окружение — 2.1.3. Как убедиться, что другие проекты не задеты

2.2. Подключение моделей и первый запуск — Как CrewAI находит модель — Наши три модели — Первый запуск и разбор скрипта — Что означает `verbose=True`

Скрипт главы: `cai_hello_crew.py`

Глава 3. Основа: Агент, Задача, Команда

3.1. Три кирпича CrewAI — 3.1.1. Агент: роль, цель, предыстория — 3.1.2. Задача: описание, результат, исполнитель — 3.1.3. Команда: сборка и запуск

3.2. Связывание задач через контекст — 3.2.1. Как одна задача видит результат другой — 3.2.2. Построение конвейера из задач

3.3. Практика: команда ревью кода — 3.3.1. Постановка задачи — 3.3.2. Как устроена команда — 3.3.3. Разбор результата

Скрипт главы: `cai_code_review_crew.py`

Глава 4. Процессы: как команда распределяет работу

4.1. Последовательный процесс — 4.1.1. Задачи по очереди — 4.1.2. Когда этого достаточно

4.2. Иерархический процесс — 4.2.1. Что меняется с появлением руководителя — 4.2.2. Распределение работы на ходу — 4.2.3. Когда нужен руководитель, а когда нет

4.3. Практика: исследование биржевого актива — 4.3.1. Постановка задачи — 4.3.2. Как устроен скрипт (+ раздел «Если доступ к SEC закрыт») — 4.3.3. Два подводных камня, которые всплыли на практике — 4.3.4. Сравнение двух режимов на живом результате

Скрипт главы: `cai_stock_research_crew.py`

Глава 5. Инструменты, знания и память

5.1. Инструменты: руки агента — 5.1.1. Зачем агенту инструменты — 5.1.2. Встроенные инструменты и свои — 5.1.3. Как устроен свой инструмент

5.2. База знаний: ответы по своим документам — 5.2.1. Зачем это нужно — 5.2.2. Как это работает внутри — 5.2.3. О пределах базы знаний

5.3. Память команды и практика: служба поддержки — 5.3.1. Что даёт память — 5.3.2. Постановка задачи — 5.3.3. Как устроен скрипт — 5.3.4. Что получилось: база знаний работает — 5.3.5. Что не получилось: память

Скрипт главы: `cai_support_crew.py`

Глава 6. Flows: точное управление процессом

6.1. Зачем нужны Flows — 6.1.1. Где упирается обычная команда — 6.1.2. Что даёт Flow — 6.1.3. Когда Flow нужен, а когда избыточен

6.2. Как устроен Flow — 6.2.1. Шаги: из чего собран процесс — 6.2.2. Переходы: как шаги связаны — 6.2.3. Состояние: общая память процесса — 6.2.4. Как это складывается вместе

6.3. Практика: поток самокоррекции кода — 6.3.1. Постановка задачи — 6.3.2. Как устроен поток — 6.3.3. Запасная модель: не лучше, а другая — 6.3.4. Что показал прогон

Скрипт главы: `cai_code_flow.py`

Глава 7. Итоги

7.1. Что мы собрали — Путь, который мы прошли — Сквозные уроки

7.2. Что удобнее в CrewAI, а что — иначе — 7.2.1. В чём выигрыш CrewAI — 7.2.2. Где готовность оборачивается ограничением — 7.2.3. CrewAI и ручная сборка процессов

7.3. Куда двигаться дальше — 7.3.1. С чего начать свою задачу — 7.3.2. На что смотреть при отладке — 7.3.3. Что осваивать следующим — 7.3.4. Главное, что стоит унести с собой

Приложения

Приложение А. Шпаргалка CrewAI — весь синтаксис курса на одной странице: обязательное начало скрипта, модель, агент, задача, команда, процессы, свой инструмент, база знаний, Flow, безопасное исполнение кода, доступ через прокси.

Приложение В. Словарь терминов — все понятия курса простыми словами: основа, процессы, модели, инструменты и знания, Flows, общее.

Приложение С. Устранение типовых проблем — реальные сбои, встреченные при прохождении, с симптомами, причинами и проверенными решениями: окружение и уста-

новка, запуск и модели, инструменты, база знаний, память, внешние источники данных, Flows, работа в PyCharm.

Скрипты курса

cai_hello_crew.py (гл.2)

Первая команда: один агент, одна задача

cai_code_review_crew.py (гл.3)

Ревью кода: автор → критик → редактор

cai_stock_research_crew.py (гл.4)

Исследование актива в двух режимах процесса

cai_support_crew.py (гл.5)

Служба поддержки по инструкции прибора (PDF)

cai_code_flow.py (гл.6)

Поток самокоррекции кода с переключением модели

Модели курса

gpt-oss:120b-cloud Основная: связный текст, рассуждения

gemma4:cloud Запасная: генерация кода, проверка

qwen2.5:7b Локальная: быстрые проверки, строгий формат

bge-m3 Локальная: поиск по документам (эмбеддинги)

Глава 1. Что такое CrewAI и зачем он нужен

1.1. Идея: команда сотрудников вместо одного агента

Кто и когда сделал CrewAI

Фреймворк создал **Жоан Моура** (João Moura) — бразильский инженер из Сан-Паулу, до этого руководивший ИИ-разработкой в компании Clearbit. Первая версия вышла в открытый доступ в декабре 2023 года и почти сразу привлекла внимание: идея «команды агентов с ролями» оказалась понятнее большинства тогдашних подходов. Сегодня CrewAI развивает одноимённая компания вместе с сообществом, код открыт и свободен для использования. Знать это полезно по одной причине: фреймворк молодой и быстро меняется, поэтому при работе всегда сверяйтесь с официальной документацией — часть деталей за год могла измениться.

Собрать агента можно вручную — например, на LangGraph: описываете узлы, соединяете их рёбрами, ведёте состояние, добавляете циклы и инструменты. Это гибко, но кропотливо: каждую связь приходится прокладывать са-

мому.

CrewAI подходит к делу с другой стороны. Вместо схемы из узлов и стрелок вы **набираете команду сотрудников**, раздаёте каждому роль и задачу — и говорите: «Работайте». Как это выглядит и когда так удобнее — разберём здесь.

Термины параграфа

— **CrewAI** — фреймворк для сборки команд ИИ-агентов. Название так и переводится: *crew* — «команда», «экипаж».

— **Агент (Agent)** — отдельный «сотрудник»: у него есть роль, цель и характер. Например, «строгий рецензент кода».

— **Задача (Task)** — конкретное поручение: что сделать и какой результат ожидается.

— **Команда (Crew)** — группа агентов, которые вместе выполняют набор задач.

Эти три слова — Агент, Задача, Команда — основа всего CrewAI. Подробно разберём их в главе 3, пока достаточно общего смысла.

Аналогия: редакция журнала

Представьте, что вам нужно выпустить статью.

Можно сделать всё в одиночку: написать черновик, самому его вычитать, самому исправить ошибки, самому оформить. Один человек, много шляп — то надеваешь шляпу автора, то шляпу редактора. Именно так работает одиночный агент: одна модель по очереди выполняет все роли.

А можно собрать **редакцию**: автор пишет, литературный редактор правит стиль, корректор ловит опечатки, выпуска-

ющий собирает всё вместе. У каждого своя зона ответственности, работа идёт по цепочке. Так устроен CrewAI: вы описываете сотрудников и их обязанности, а передачу дел между ними берёт на себя фреймворк.

Отсюда главная мысль: **разделение ролей часто даёт результат лучше, чем один исполнитель на всё**. Специалист, сосредоточенный на одной задаче, ошибается реже, чем универсал, который держит в голове сразу всё.

Этот образ — отдел с распределёнными ролями — пройдёт через весь курс. Держите его в голове: почти всё устройство CrewAI на него ложится.

1.1.1. Чем CrewAI отличается от ручной сборки на LangGraph

В LangGraph вы управляете процессом **на уровне механики**: вот узел, вот переход, вот условие ветвления. Вы решаете буквально каждый шаг — полный контроль ценой ручной работы.

В CrewAI вы работаете **на уровне намерения**: описываете, *кто* эти сотрудники и *что* нужно сделать, а как передать работу от одного к другому — решает фреймворк.

Коротко разница такая:

— **LangGraph** — вы конструктор процесса. Максимум контроля, максимум ручной работы.

— **CrewAI** — вы руководитель команды. Быстрее собрать, но часть решений принимает фреймворк за вас.

Ни один подход не «лучше» вообще — они хороши для

разного. К этому вернёмся в пункте 1.1.3.

1.1.2. Метафора «отдел с ролями»

Разложим образ редакции на составные части — они и есть три главных понятия CrewAI:

— **Агент — это сотрудник.** У него есть должность (роль), цель работы и профессиональный опыт (предыстория, которая задаёт манеру поведения).

— **Задача — это поручение в работу.** С понятным описанием и ожидаемым результатом, закреплённое за конкретным сотрудником.

— **Команда — это сам отдел.** Она объединяет сотрудников и поручения и определяет, в каком порядке всё делается.

Когда в главе 3 вы будете писать первый код, вы буквально опишете сотрудников и их поручения обычными словами. CrewAI силён тем, что настройка агента близка к человеческому языку: «ты — опытный рецензент, твоя цель — найти слабые места в коде, ты дотошен и не пропускаешь мелочей».

1.1.3. Когда CrewAI удобнее, а когда лучше LangGraph

Границы применимости важнее списка достоинств.

CrewAI удобнее, когда:

— задача естественно делится на роли (автор — критик — редактор);

— нужно быстро собрать рабочий прототип команды, не выписывая всю схему вручную;

— шаги в целом идут по порядку или их распределяет агент-руководитель;

— вас устраивает, что часть решений о ходе работы фреймворк примет сам.

LangGraph лучше, когда:

— нужен точный контроль над каждым переходом и ветвлением;

— логика сложная: много условий, циклов, возвратов на предыдущие шаги;

— важно предсказуемо управлять состоянием и точками остановки (например, вмешательство человека в конкретный момент);

— вы хотите видеть и настраивать весь маршрут процесса, а не доверять его фреймворку.

Есть и золотая середина: у CrewAI есть механизм **Flows** (глава 6), который возвращает точное управление порядком шагов — уже поверх команд. По духу он ближе к LangGraph. То есть выбор не всегда «или-или»: инструменты можно сочетать, и в главе 7 мы разберём, как именно.

Главное, что стоит унести из этого параграфа: **CrewAI — это про описание команды словами, LangGraph — про черчение процесса руками.** Разные уровни управления под разные задачи.

1.2. Какие задачи решает — примеры из жизни

Мы договорились: CrewAI хорош там, где работа естественно делится на роли. Посмотрим на три живых примера — на них видно, где командный подход раскрывается в полную силу. Пока только на уровне идеи, без кода; ко всем трём задачам мы вернёмся практически в следующих главах.

Термины параграфа

— **Роль (role)** — «должность» агента, одной фразой: кто он и за что отвечает.

— **Конвейер** — цепочка, где результат одного сотрудника становится сырьём для следующего.

1.2.1. Команда ревью кода: автор, критик, редактор

Проблема из жизни. Вы написали функцию и хотите, чтобы её проверили: нет ли ошибок, всё ли понятно, можно ли улучшить. В одиночку это неудобно — трудно быть одновременно и автором, и придирчивым проверяющим. Глаз замыливается: свои огрехи не видишь.

Как это ложится на команду. Разбиваем работу на три роли:

— **Автор** — пишет первую версию кода по заданию.

— **Критик** — придирчиво читает и перечисляет проблемы: ошибки, неясные места, слабые решения.

— **Редактор** — берёт замечания критика и выпускает ис-

правленную, чистовую версию.

Работа идёт конвейером: автор → критик → редактор. Каждый видит результат предыдущего. Это ровно та редакция, о которой мы говорили в 1.1.

Почему команда здесь уместна. Роли очевидны и не пересекаются, порядок работы естественный. Критику проще быть строгим, когда он не отвечает за написание, а только ищет слабые места. Классический случай, где разделение ролей улучшает результат. Соберём такую команду в главе 3.

1.2.2. Исследование темы и подготовка отчёта

Проблема из жизни. Нужно разобраться в теме и сделать короткую сводку: собрать факты, отсеять лишнее, изложить понятно. Одному агенту тяжело удержать всё сразу — он и ищет, и оценивает, и пишет, и на каком-то шаге обычно «съезжает».

Как это ложится на команду.

— **Исследователь** — собирает материал по теме (при необходимости — с поиском в интернете как инструментом).

— **Аналитик** — отбирает из собранного главное, отбрасывает шум.

— **Автор отчёта** — превращает отобранное в связный, читаемый текст.

Здесь появляется важная деталь: работу можно организовать не жёсткой цепочкой, а с **агентом-руководителем**, который сам решает, кому что поручить и в каком порядке.

Такой способ распределения называется иерархическим — разберём его в главе 4.

Почему команда здесь уместна. Задача крупная и делится на явные этапы. Разделив её, мы не даём одной модели захлебнуться объёмом. Плюс это удобный случай показать инструменты (поиск) и руководителя.

1.2.3. Многошаговые процессы с несколькими исполнителями

Проблема из жизни. Реальная работа редко уместается в один шаг. Пример: собрать свежие новости по теме → отфильтровать важное → написать короткий дайджест → проверить, что он читается. Шагов много, у каждого свой характер.

Как это ложится на команду. Каждый шаг закрепляется за своим исполнителем со своей ролью. Но чем длиннее цепочка, тем сильнее хочется управлять ею точнее: а что, если проверка на последнем шаге забраковала дайджест — вернуть на доработку? Простой конвейер такого разворота назад не умеет.

Вот здесь заканчиваются возможности обычной команды и начинаются **Flows** (глава 6) — механизм, который позволяет задавать порядок шагов явно, с ветвлениями и возвратами.

Почему это важно понять сразу. Чтобы не было завышенных ожиданий. Обычная команда CrewAI хорошо справляется с линейной или веерной работой, но сложные марш-

руты с условиями — уже уровень Flows. Мы пройдем и то, и другое.

Итог трёх примеров: как только задача делится на роли, её удобно решать командой. Ревью кода — простой конвейер из трёх ролей. Исследование — команда с руководителем, который раздаёт работу. Многошаговые процессы с проверками и возвратами — забота Flows. Все три задачи мы соберём своими руками в следующих главах.

1.3. Карта курса

Прежде чем браться за инструмент, полезно увидеть весь маршрут: куда идём и какие остановки нас ждут. Держите эту карту под рукой: если в середине курса покажется, что вы заблудились, вернитесь сюда и посмотрите, где находитесь.

1.3.1. Что освоим по главам

Курс устроен как подъём по ступеням — от простого к сложному. Каждая следующая глава опирается на предыдущую.

— **Глава 2. Подготовка окружения.** Поставим CrewAI в отдельное виртуальное окружение и подключим модели через Ollama. Здесь же разберём практический урок: почему CrewAI держат в изолированном окружении.

— **Глава 3. Основа: Агент, Задача, Команда.** Три главных кирпича. Соберём первую рабочую команду — ревью кода из трёх ролей.

— **Глава 4. Процессы.** Как команда распределяет работу: по очереди (последовательно) или через агента-руководителя (иерархически). Соберём исследование биржевого актива и прогоним его в обоих режимах.

— **Глава 5. Инструменты, знания, память.** Дадим агентам «руки» (инструменты), подключим базу знаний из документов и разберём память команды. Соберём службу поддержки, отвечающую по инструкции прибора.

— **Глава 6. Flows.** Точное управление сложным процессом: явный порядок шагов, ветвления, возвраты, состояние. Соберём поток самокоррекции кода. Это самая мощная часть CrewAI и самая близкая по духу к LangGraph.

— **Глава 7. Итоги.** Соберём общую картину и разберём практический вопрос: что удобнее делать в CrewAI, что — другими средствами, и как их сочетать.

— **Приложения.** Шпаргалка на один лист, словарь терминов и разбор типовых проблем — чтобы возвращаться за справкой после курса.

Логика движения простая: сначала научимся собирать команду, потом управлять тем, как она работает, потом расширять её возможности, и в конце — брать процесс под полный контроль.

1.3.2. Возможности CrewAI одним взглядом

Короткий обзор того, что умеет фреймворк. Слева — возможность, справа — зачем она нужна. Пока просто прочитайте, чтобы держать общую картину.

Возможность — Зачем нужна

Агент (Agent) с ролью и целью — Задать поведение «сотрудника» обычными словами

Инструменты (Tools) — Дать агенту «руки»: поиск, работа с файлами, вычисления

Задачи (Tasks) и порядок их выполнения — азложить работу на понятные поручения

Процессы (последовательный, иерархический) —

Выбрать, как команда распределяет работу

База знаний (Knowledge RAG) — Отвечать по своим заданным документам без ручной сборки поиска

Память команды — Помнить факты и ход работы между шагами и запусками

Flows — Управлять сложным процессом явно: ветвления, возвраты, состояние
Вмешательство человека
Остановиться и спросить человека в нужный момент

Первые семь строк таблицы мы разберём подробно, каждую в своей главе. Последняя — вмешательство человека — в курс не входит; о ней скажем в главе 7 как о направлении для самостоятельного изучения.

Общий вывод: **CrewAI** даёт готовые узлы там, где иначе пришлось бы собирать механизм из деталей вручную. Ваша задача на этом курсе — понять, где такая готовность экономит силы, а где полезнее взять управление на себя.

Итог главы 1

— CrewAI собирает работу из трёх кирпичей — **Агент, Задача, Команда** — и организует её как работу отдела с распределёнными ролями.

— В отличие от LangGraph, где процесс чертят вручную, здесь команду **описывают словами**, а передачу дел берёт на себя фреймворк.

— Это удобно, когда задача делится на роли; для сложных

маршрутов с условиями есть **Flows**.

— Почти всё для сильных команд агентов у CrewAI есть «из коробки».

Теория закончилась. В главе 2 подготовим рабочее окружение и сделаем первый запуск.

Глава 2. Подготовка окружения

2.1. Отдельное окружение под CrewAI

Прежде чем писать первого агента, нужно подготовить рабочее место. Мы проводим тесты под управлением OS Ubuntu24.04 линукс пожалуй самая удобная среда для работы с ИИ, но возможна работа и в других операционных системах с некоторыми отличиями комнд. Если вы используете иную операционную систему то в среде разработки PyCharm большинство интрукций не зависят от OS. В ходе подготовки окружения в других OS могут возникнуть вопросы, которые легко решить с помощью любого ИИ чата, просто показав ему текст ошибки или задав вопрос по установке пакета. Звучит скучно, но именно здесь новички теряют больше всего времени: ставят библиотеку не туда, ломают уже работающий проект и потом полдня ищут причину.

Главная мысль параграфа: **CrewAI ставят в отдельное, изолированное окружение.** Сейчас объясним почему и сделаем это.

Термины параграфа

— **Виртуальное окружение (venv)** — отдельная «коробка» с Python и библиотеками под один проект. То, что лежит в одной коробке, не влияет на другие.

— **Зависимость** — сторонняя библиотека, которая нужна вашему проекту для работы. У самих библиотек тоже есть свои зависимости.

— **Конфликт зависимостей** — ситуация, когда двум библиотекам нужна одна и та же третья, но **разных, несовместимых версий**. Одно окружение не может держать две версии сразу — приходится выбирать, и кто-то остаётся сломанным.

2.1.1. Почему CrewAI нужно свое окружение

Дело в общей библиотеке по имени **chromadb** — это хранилище для поиска по документам. От неё зависят сразу несколько инструментов, но **каждому может быть нужна своя версия**:

— CrewAI тянет за собой определённую линейку chromadb;

— библиотеки экосистемы LangChain (например, langchain-chroma) требуют другую версию той же chromadb.

Поставьте оба в одно окружение — и менеджер пакетов встанет перед неразрешимой задачей: держать две несовместимые версии одной библиотеки одновременно он не может. В лучшем случае установка оборвётся с ошибкой. В худшем — молча поставит одну версию, и **сломается то, что раньше работало**: другой рабочий проект вдруг начнёт падать, хотя вы его не трогали.

Запомните этот случай как общее правило: **проекты с**

конфликтующими зависимостями держат в разных окружениях. Не пытайтесь ужать всё в одно «чтобы не плодить папки» — сэкономите место, потеряете дни на отладку.

2.1.2. Создаём проект и окружение

Что нужно иметь до начала:

— **Python 3.10 или новее.** CrewAI на более старых версиях не работает.

— **Установленную и запущенную Ollama.** Через неё мы подключим модели.

— **Установленный PyCharm** (мы весь курс работаем именно в нём), окружение проще создать средствами среды:

— *File* → *New Project*, имя проекта — `crew_ai_course`.

— В настройках интерпретатора выберите **New environment** типа *Virtualenv* и Python 3.10 или новее. PyCharm сам создаст папку `venv` внутри проекта.

— Откройте встроенный терминал (*View* → *Tool Windows* → *Terminal*). PyCharm активирует нужное окружение автоматически — в начале строки появится `(.venv)`.

Если предпочитаете командную строку, то же самое делается вручную:

```
mkdir ~/PycharmProjects/crew_ai_course
```

```
cd ~/PycharmProjects/crew_ai_course
```

```
python3 -m venv. venv
```

```
source. venv/bin/activate
```

Признак успеха один и тот же: приглашение командной строки начинается с `(.venv)` — вы «на нужной кухне».

Дальше все команды выполняем в этом терминале. Ставим сам CrewAI и то, что понадобится по ходу курса:

```
pip install — upgrade pip
```

```
pip install crewai
```

```
pip install yfinance requests «requests [socks]» ddgs ollama
```

Что мы поставили и зачем:

— **crewai** — сам фреймворк. Вместе с ним подтягиваются агенты, задачи, команды, потоки и работа с базой знаний;

— **yfinance** — источник биржевых данных, понадобится в главе 4;

— **requests** — обращения к внешним сервисам из наших инструментов; дополнение requests [socks] нужно, если доступ к сайту американского регулятора у вас закрыт и придётся идти через прокси (об этом в главе 4);

— **ddgs** — поиск в интернете, на нём построен инструмент новостей в главе 4;

— **ollama** — клиент к Ollama на стороне Python. Он нужен базе знаний в главе 5 для поиска по документам; сам CrewAI его отдельно не тянет, поэтому ставим руками.

Установка занимает несколько минут: CrewAI тянет за собой большое дерево зависимостей. Это нормально.

Проверяем, что фреймворк встал:

```
python -c «import crewai; print (crewai.__version__)»
```

Если версия напечаталась — окружение готово.

2.1.3. Как убедиться, что другие проекты не задеты

Простая гигиена, которая избавит от тревоги: пока вы

работаете с CrewAI — активно только окружение `crew_ai_course`. Другие ваши проекты лежат в своих папках, со своими окружениями, и мы туда ничего не устанавливаем.

Проверить, где вы находитесь, можно одной командой:
`which python`

Она покажет, какой Python сейчас активен. Если путь ведёт в `crew_ai_course/.venv/bin/python` — вы там, где нужно. Если путь другой (например, `/usr/bin/python3`), окружение не активировано: вернитесь к команде `source .venv/bin/activate`.

В PyCharm то же самое видно без команд: текущий интерпретатор показан в правом нижнем углу окна и в *File* → *Settings* → *Project* → *Python Interpreter*. Убедитесь, что там выбрано окружение проекта `crew_ai_course`, а не системный Python.

Правило на весь курс: **сначала убедись, что активно окружение CrewAI, потом ставь пакеты**. Две секунды проверки экономят часы разбора поломок.

Окружение готово. В следующем параграфе подключим модель и запустим первую команду.

2.2. Подключение моделей и первый запуск

дочитайте главу до конца и затем переходите к работе с файлом примера в Pycharm.

(!) В коде всех примеров имеются параметры:

у агента `verbose=True`

у команды `verbose=True`

Эти параметры отвечают за вывод промежуточных итогов внутреннего взаимодействия команды, по умолчанию они стоят в `True` что бы вы могли видеть что делают ваши агенты и последовательность их действий.

Вы можете устанавливать эти значения в `False` тогда CrewAI не будет выдавать наружу свои внутренние процессы, вы увидите только результат.

Окружение готово, CrewAI установлен. Пора оживить команду — подключить модель и сделать первый запуск. Разберём, как CrewAI разговаривает с моделями через Ollama, какие модели мы используем и почему, и по частям поймём устройство первого скрипта.

Термины параграфа

— **Ollama** — программа, которая запускает языковые модели у вас на машине (локально) или даёт доступ к облачным.

— **LiteLLM** — «переходник» между CrewAI и разными

поставщиками моделей. Благодаря ему CrewAI одинаково обращается и к OpenAI, и к Ollama — меняется только имя модели.

— **Префикс ollama/** — приставка к имени модели, которая говорит LiteLLM: «эту модель бери через Ollama».

— **kickoff ()** — команда «начать работу». Именно она запускает всю команду агентов.

Как CrewAI находит модель

CrewAI сам не умеет говорить с моделями напрямую — он поручает это LiteLLM. А LiteLLM понимает десятки поставщиков и различает их по **приставке к имени модели**.

Аналогия: международный телефонный звонок. Чтобы дозвониться, перед номером набирают код страны. ollama/ — такой же код: «соединить через Ollama». Пишете ollama/gpt-oss:120b-cloud — и LiteLLM понимает, что за моделью надо идти в Ollama.

Забудете префикс — LiteLLM решит, что это модель другого поставщика, и либо потребует ключ, либо выдаст ошибку. Это одна из самых частых оплошностей новичка, поэтому запомните правило: **для Ollama имя модели всегда начинается с ollama/**.

Наши три модели

В курсе работаем с таким набором:

— **Основная — gpt-oss:120b-cloud**. Мощная облачная модель, доступная через Ollama (на момент написания бесплатно). На ней держим большинство задач.

— **Запасная** — **gemma4:cloud**. Тоже облачная. Пригодится, если основная недоступна или капризничает. На задачах генерации кода она нередко показывает себя лучше — это пригодится в главе 6.

— **Локальная** — **qwen2.5:7b**. Небольшая модель, работает целиком на вашей машине без интернета. Хороша для быстрых проверок и там, где нужен строгий формат ответа для нее требуется видеокарта с памятью $\geq 8\text{Gb}$ (если у вас нет такой видеокарты то просто указывайте вместо этой модели любую из бесплатных облачных например **gemma4:cloud**).

Выбор модели в CrewAI меняется в одну строчку — именем в настройке LLM. Это удобно: можно собрать команду на основной модели, а для одного «тяжёлого» агента подставить другую.

Первый запуск

Скрипт **cai_hello_crew.py** — самая маленькая возможная команда: один сотрудник, одно поручение. Положите его в папку проекта **crew_ai_course** и запустите.

В PyCharm достаточно открыть файл и нажать зелёную стрелку запуска (или *Ctrl+Shift+F10*). Из терминала — обычной командой:

```
python cai_hello_crew.py
```

В консоли пойдут рамки с ходом работы, а в конце — блок «РЕЗУЛЬТАТ» с ответом агента. Если это произошло, значит установка модулей и окружения прошла успешно и мо-

дель отвечает вам.

Разбор скрипта

Откройте `cai_hello_crew.py` — разберём его по частям.
Это скелет, который вы будете наращивать весь курс.

Конец ознакомительного фрагмента.

Текст предоставлен ООО «Литрес».

Прочитайте эту книгу целиком, [купив полную легальную версию](#) на Литрес.

Безопасно оплатить книгу можно банковской картой Visa, MasterCard, Maestro, со счета мобильного телефона, с платежного терминала, в салоне МТС или Связной, через PayPal, WebMoney, Яндекс.Деньги, QIWI Кошелек, бонусными картами или другим удобным Вам способом.