

Обзор интегрированной среды разработки PyCharm

Работая в группе с Лебедевой Н. Г., я выполнила обзор следующей части функций среды для разработки на Python PyCharm: отладка, запуск, выполнение и сборка проекта, версионирование, публикация в репозитории, AI-функции. В соответствии с ниже следующим отчетом мною были выполнены слайды презентации с 11 по 19.

Версия: PyCharm 2025.2.2

Отладка

Отладка – один из ключевых инструментов для анализа и исправления кода. В PyCharm предусмотрено несколько разновидностей точек останова, каждая из которых используется для определённых целей.

- Обычная точка останова останавливает выполнение программы в указанной строке кода. Она используется в тех случаях, когда необходимо пошагово проанализировать выполнение программы, просмотреть значения переменных и определить, на каком этапе возникает ошибка. Обычная точка останова ставится нажатием левой кнопки мыши на номер строки слева от нее самой.
- Условная точка останова срабатывает только тогда, когда выполняется заданное пользователем логическое условие. Например, можно указать, что программа должна приостановиться только если переменная принимает определённое значение. Такой подход позволяет не останавливать выполнение лишний раз, а сосредоточиться на проблемных случаях. При нажатии правой кнопкой мыши на точку останова откроется меню, в котором можно написать условие в графе «Condition», активировав перед этим соответствующую галочку.
- Точка останова «только лог» не останавливает выполнение кода, а выполняет действие, например, выводит сообщение в консоль, записывает значение переменной или стек вызовов. Это полезно, когда требуется отследить ход работы программы, не прерывая её выполнения. Поставить такую точку можно, деактивировав опцию «Suspend», о которой подробнее будет написано ниже.

Таким образом, разные виды точек останова дают возможность настроить отладку максимально гибко: от базового пошагового анализа до

тонкой фильтрации и логирования, что значительно ускоряет процесс поиска и устранения ошибок.

При настройке точки останова в PyCharm предусмотрена опция Suspend, которая определяет, будет ли программа останавливаться при её срабатывании. Если галочка активна, выполнение приостанавливается в заданной строке. Если же снять галочку, то точка останова может использоваться только для дополнительных действий, например вывода сообщений в консоль или логирования значений переменных, не прерывая при этом работы программы.

Дополнительно можно выбрать режим приостановки: All или Thread. В первом случае при срабатывании точки останова останавливаются все потоки выполнения программы, что удобно при анализе общего состояния приложения. Во втором случае приостанавливается только тот поток, в котором сработала точка останова, а остальные продолжают работу. Это позволяет исследовать отдельный поток без блокировки всей программы.

Существуют еще некоторые параметры точки останова:

- "Breakpoint hit" message – при срабатывании точки останова в консоль будет автоматически выводиться сообщение о том, что она достигнута. Это полезно для наглядного отслеживания, в каком месте программы произошла остановка.
- Stack trace – вместе с сообщением выводится полный стек вызовов (последовательность функций, которые привели к этой строке). Такой вариант помогает понять контекст выполнения программы, особенно если точка останова находится глубоко внутри вложенных функций.
- Evaluate and log – позволяет указать выражение, которое будет вычисляться и выводиться в лог при каждом срабатывании точки останова. Например, можно автоматически записывать текущее значение переменной или результат вычислений.
- Remove once hit – точка останова автоматически удаляется сразу после первого срабатывания. Такой вариант удобен, если нужно отследить только первое прохождение кода и не прерывать выполнение в дальнейшем.

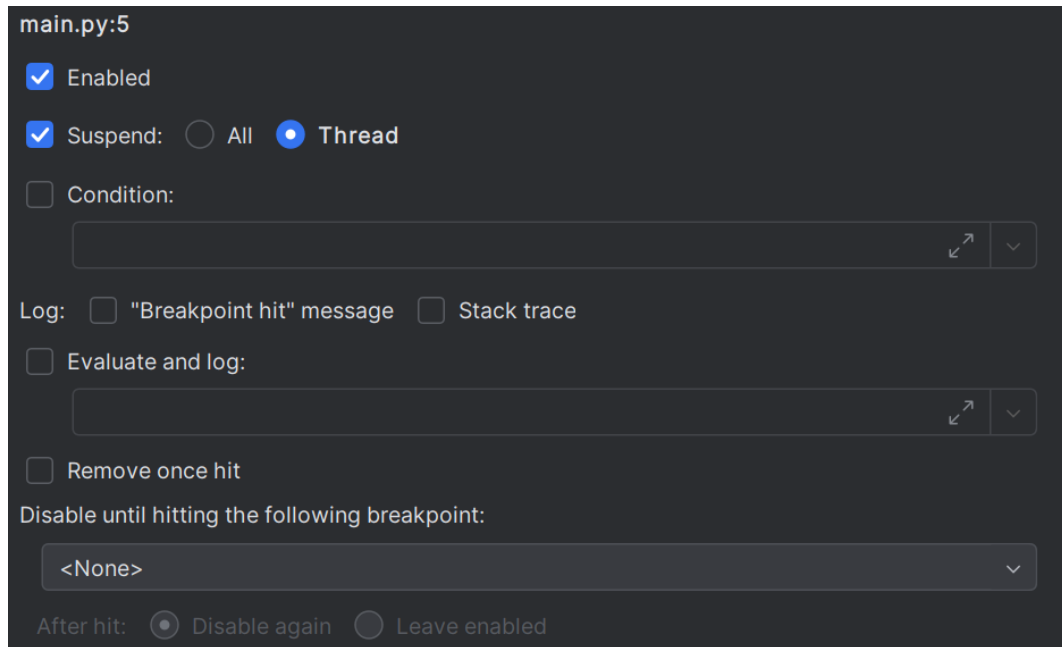


Рисунок 1 – Окно настроек точки останова

После остановки исполнения становится доступна специальная панель, на которой можно просмотреть значения переменных (Variables) – в правой части панели, текущее состояние стека вызовов (Frames) – левая часть панели, а также задать дополнительные наблюдения (Watches). Для детального анализа предусмотрено пошаговое выполнение: переход по строкам кода (Step Over), заход внутрь функций (Step Into), выход из функции (Step Out) и особый режим Step Into My Code, который игнорирует сторонние библиотеки – все эти функции доступны при нажатии на соответствующую иконку со стрелочкой (в верхней части панели), название функции можно увидеть при наведении курсора. При необходимости можно использовать команду Force Run to Cursor, чтобы ускорить выполнение программы до выбранного места, команда доступна при нажатии на три точки в той же строке с иконками, либо при нажатии Ctrl+Alt+F9.

Отдельно стоит отметить функцию Evaluate Expression (Alt+F8). Она позволяет в любой момент выполнения выполнить произвольное выражение, проверить работу фрагмента кода или вывести интересующее значение. Эта возможность особенно полезна для проверки гипотез или быстрой отладки без необходимости изменять основной код. Автодополнение и подсветка синтаксиса при этом сохраняются, что делает работу с функцией удобной и интуитивно понятной. Этой функцией также можно воспользоваться в правой панели (где написано «Evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)»). В это поле можно вручную добавить выражения, за

которыми нужно следить постоянно, даже если они не являются переменными текущей области видимости.

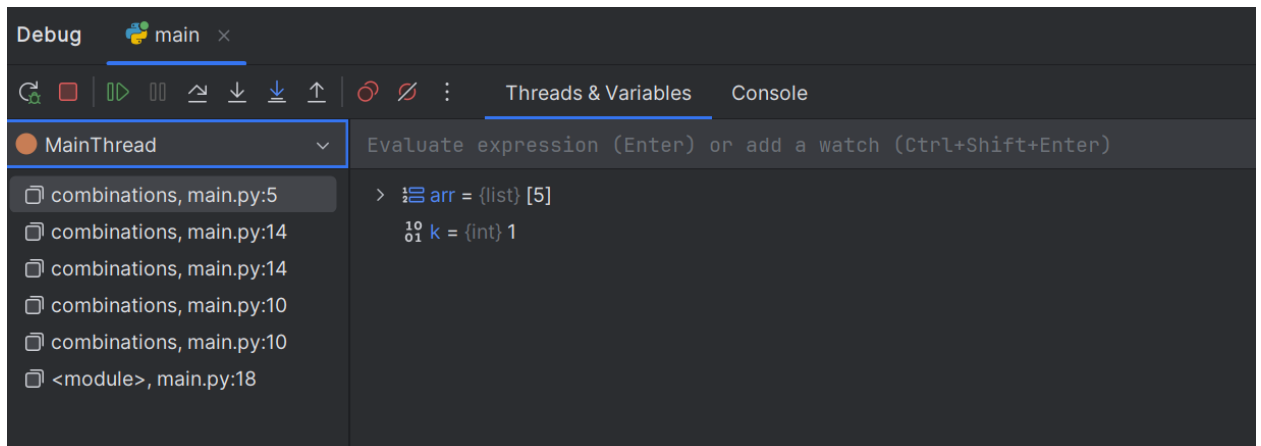


Рисунок 2 – Панель отладки с отображением стека вызовов и переменных

Запуск программы

Запуск программ в PyCharm организован через механизм Run/Debug Configurations, который позволяет управлять параметрами выполнения. Для каждого сценария можно задать собственную конфигурацию. В настройках конфигурации указываются интерпретатор Python, аргументы командной строки, переменные окружения и рабочая директория. При необходимости можно объединить несколько конфигураций в так называемый compound-запуск, чтобы они выполнялись последовательно или одновременно.

На скриншоте (рисунок 3) представлено окно создания и редактирования конфигурации запуска в PyCharm. В верхней части задаётся Name – имя конфигурации. Оно может быть любым и используется для удобства, чтобы различать разные сценарии запуска в выпадающем списке.

Ниже находится выпадающий список выбора интерпретатора Python (в данном случае Project Default – Python 3.10). Здесь определяется, в какой среде будет выполняться программа: можно выбрать интерпретатор по умолчанию, виртуальное окружение или системный Python.

Следующее поле отвечает за тип запуска – в примере указан вариант *script*, то есть выполнение конкретного файла `main.py`. При необходимости можно переключить на запуск модуля или пакета. В соседней строке указывается путь к исполняемому файлу.

Опция Script parameters позволяет передать аргументы командной строки. Это удобно, если программа принимает параметры при запуске.

Поле Working directory определяет рабочую директорию – от неё будет зависеть разрешение относительных путей в коде. В примере задана папка проекта `chernovik`.

Далее можно задать Environment variables – переменные окружения, которые будут действовать во время запуска программы. В примере установлено значение `PYTHONUNBUFFERED=1`, что отключает буферизацию вывода в консоль.

Ниже предусмотрено поле для указания пути к файлу `.env`, если в проекте используются переменные окружения в таком формате. Это позволяет автоматически подгружать их при запуске.

Ниже располагаются дополнительные параметры:

- Open run/debug tool window when started – автоматически открывать панель отладки и запуска при старте программы.
- Add content roots to PYTHONPATH – добавлять корневые каталоги проекта в переменную `PYTHONPATH`, чтобы модули проекта можно было импортировать без ошибок.
- Add source roots to PYTHONPATH – аналогично предыдущему, но применяется к каталогам исходного кода.

Таким образом, это окно позволяет настроить запуск программы: выбрать интерпретатор и рабочую директорию, задать параметры, переменные окружения и дополнительные опции, что делает работу с проектом удобной и воспроизводимой.

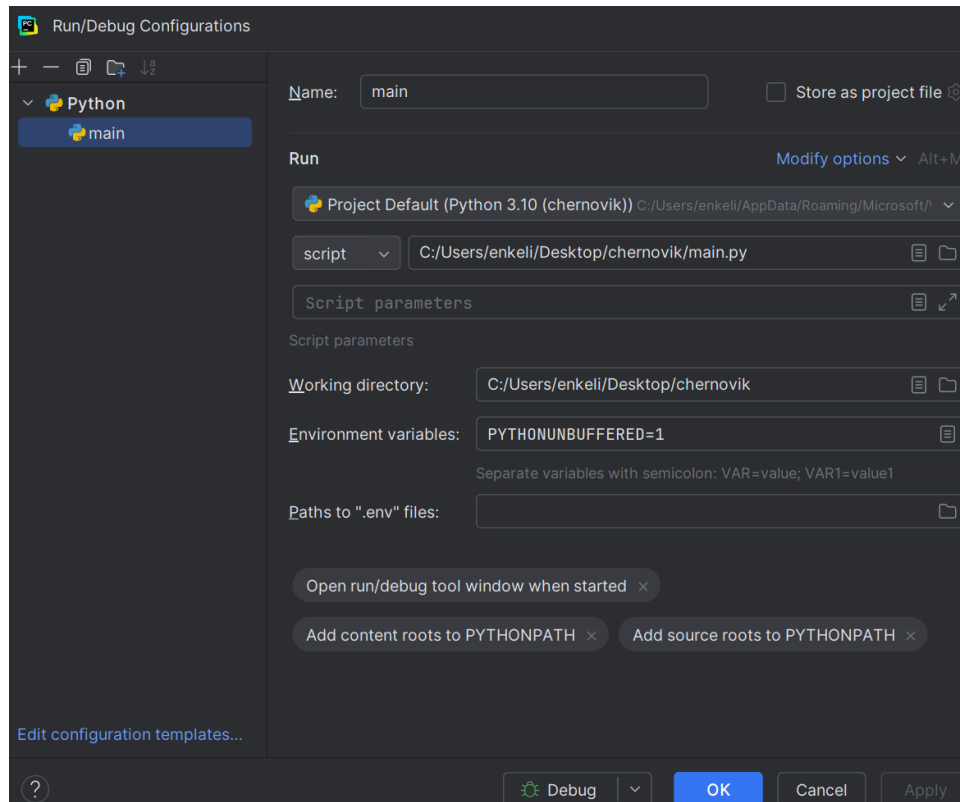


Рисунок 3 – Окно настроек Run/Debug Configurations

Выполнение и сборка проекта

В отличие от компилируемых языков программирования (например, C++ или Java), Python является интерпретируемым языком. Это означает, что при запуске программы исходный код построчно обрабатывается интерпретатором Python, а не преобразуется заранее в исполняемый файл. Поэтому в классическом смысле компиляции в PyCharm нет.

Тем не менее, интерпретатор Python при первом импорте модуля автоматически преобразует код в байткод и сохраняет его в виде файлов с расширением `.pyc` в папке `__pycache__`. Эти файлы служат кешем и ускоряют последующие запуски программы, но они полностью управляются самим интерпретатором, а не средой разработки.

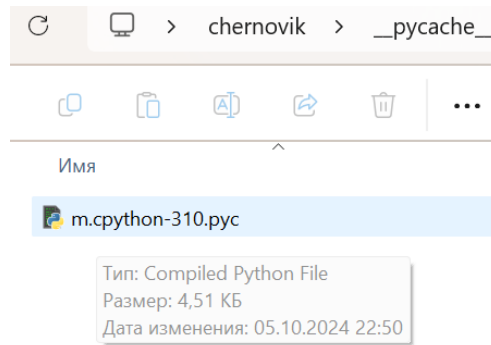


Рисунок 4 – Файл байткода .рус в папке `__pycache__`, созданный интерпретатором Python

Что касается PyCharm, то здесь под «сборкой» проекта чаще всего подразумевается упаковка программы в специальный формат для распространения: например, wheel или sdist. Для этого используются стандартные инструменты Python (setuptools, poetry, pdm), а PyCharm предоставляет интерфейс для их настройки и запуска. Таким образом, IDE помогает подготовить проект к публикации, но не выполняет компиляцию кода в привычном смысле этого слова.

Версионирование и публикация в репозитории

В PyCharm предусмотрена интеграция с системами контроля версий, прежде всего с Git, но также поддерживаются Mercurial и Subversion. Среда позволяет выполнять полный цикл работы с репозиторием: инициализацию, фиксацию изменений, редактирование коммитов (amend), а также выборочную подготовку файлов к коммиту по отдельным изменениям (staging по чанкам). Для анализа доступны история изменений и просмотр авторства строк (blame), что облегчает отслеживание, кто и когда вносил правки.

Через меню Git доступны все стандартные команды. Если нажать на Commit, то откроется отдельная панель (в левой части окна на скриншоте, рисунок 5), в которой можно увидеть произошедшие изменения и неотслеживаемые файлы, написать сообщение к коммиту и совершить его.

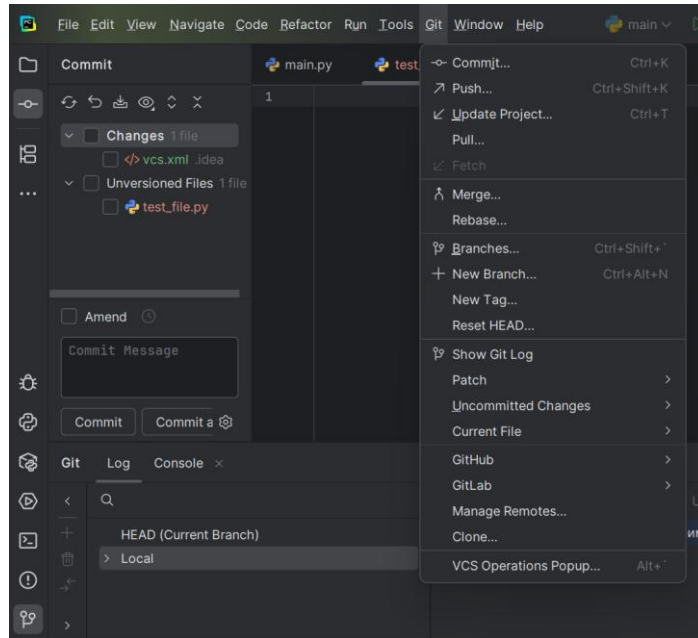


Рисунок 5 – Меню Git и панель для создания коммита

Также все основные действия можно вызывать через *VCS Operations Popup* (Alt+').

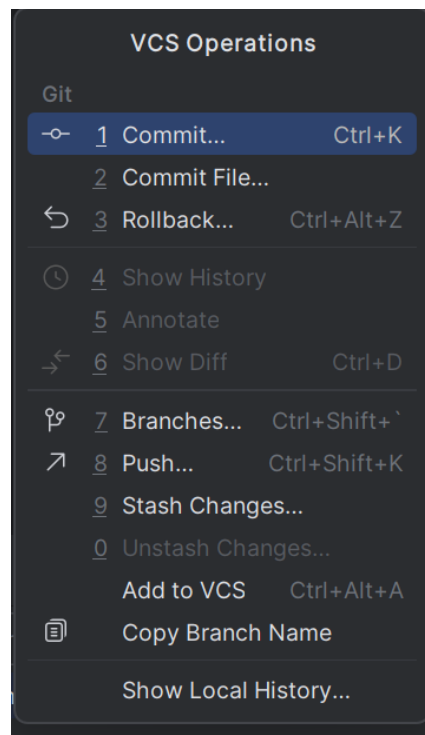


Рисунок 6 – панель VCS Operations Popup

На скриншоте (рисунок 7) представлены основные операции с ветками.

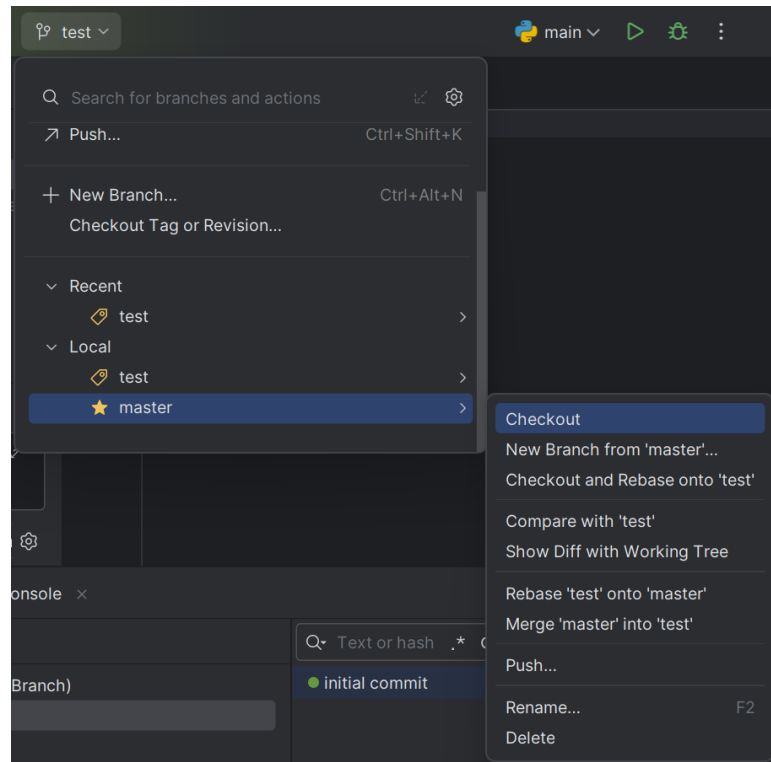


Рисунок 7 – Работа с ветками через меню Branches

А в нижней части окна доступен журнал коммитов: можно увидеть комментарий к коммиту, ветку, автора и время. При нажатии на коммит в правой части панели отображается информация о выбранном коммите: изменения и комментарий.

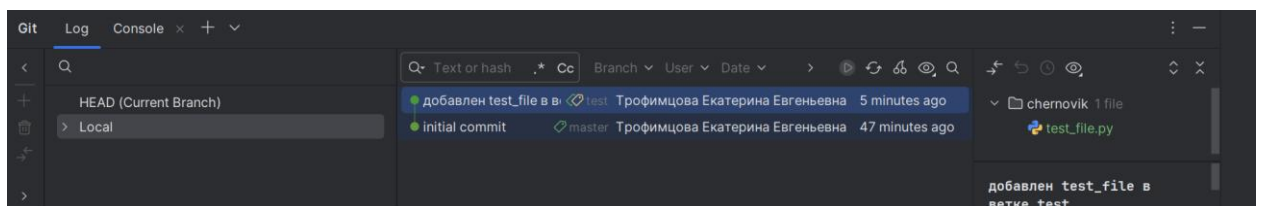


Рисунок 8 – Журнал коммитов

PyCharm поддерживает прямую публикацию проектов в удалённые репозитории, такие как GitHub и GitLab. Для этого в IDE предусмотрена команда Share Project on GitHub/GitLab, которая позволяет загрузить локальный проект в новый репозиторий на GitHub/GitLab буквально в несколько шагов. После выполнения первого коммита можно сразу выполнить Share/Push, и при необходимости PyCharm автоматически создаст удалённый репозиторий.

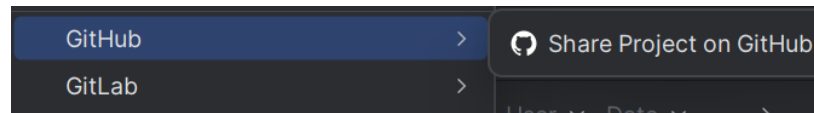


Рисунок 9 – Публикация проекта на GitHub через команду Share Project

Таким образом, PyCharm предоставляет удобный и полный набор инструментов для управления версиями, публикации и совместной разработки, позволяя разработчику работать с кодом и управлять репозиторием, не покидая среды разработки.

AI-функции

Чтобы пользоваться AI-функциями от JetBrains, нужно для начала войти в свой аккаунт JetBrains или зарегистрироваться через Google, GitHub, GitLab или Bitbucket.

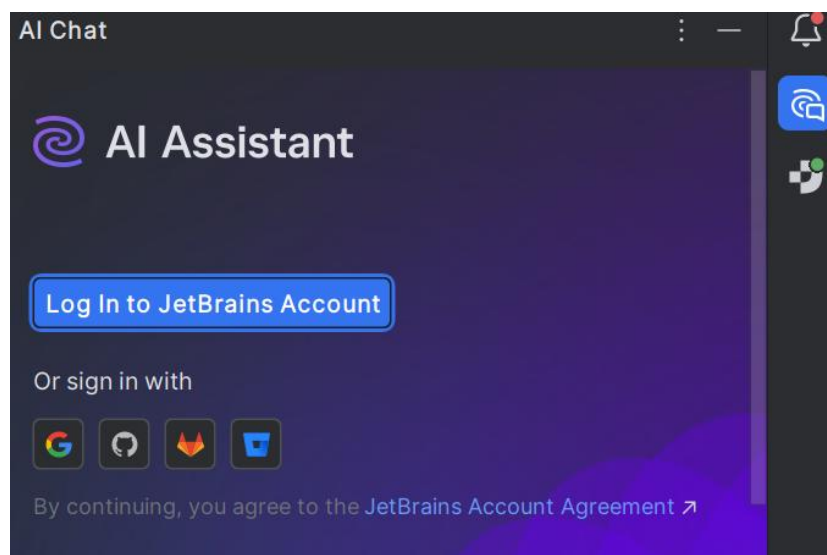


Рисунок 10 – Окно входа в AI Assistant

К сожалению, воспользоваться функциями ИИ-помощника не удалось, так как требуется привязка банковской карты к аккаунту. Однако, прямо в PyCharm и в Интернете можно прочитать, что предлагает ИИ-ассистент от JetBrains.

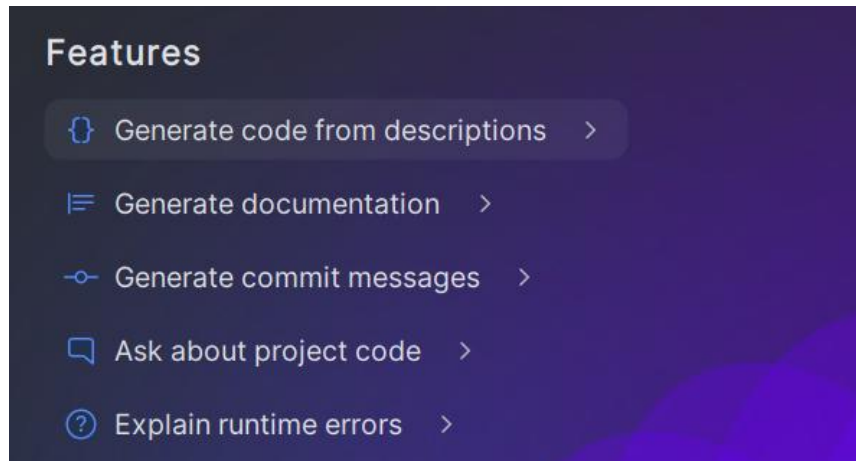


Рисунок 11 – Основные функции AI Assistant от JetBrains

- Generate code from descriptions – позволяет создавать фрагменты кода по текстовому описанию. Пользователь формулирует задачу, а ассистент предлагает готовый вариант реализации. Это особенно удобно для написания функций, классов или вспомогательных блоков кода.
- Generate documentation – формирует документацию к функциям, классам и модулям, основываясь на сигнатуре и их содержимом. Это экономит время и помогает поддерживать единый стиль документации в проекте.
- Generate commit messages – автоматически создаёт сообщение для коммита на основе изменений в коде.
- Ask about project code – функция, позволяющая задавать вопросы о проекте прямо в IDE. Ассистент анализирует код и даёт пояснения: как устроен конкретный модуль, как работает функция или где используется переменная. Это ускоряет понимание чужого кода и облегчает адаптацию к новым проектам.
- Explain runtime errors – объясняет ошибки, возникающие во время выполнения программы. Ассистент анализирует текст ошибки, указывает на её причины и предлагает возможные пути решения.

Таким образом, AI-ассистент превращает PyCharm в полноценную интеллектуальную среду, которая не только выполняет роль редактора, но и активно помогает в написании, документировании и отладке кода.

Список источников

1. Пробуем Junie от JetBrains на реальной задаче (или как я попал в рассказ Азимова). — Текст : электронный // Хабр : [сайт]. — URL: <https://habr.com/ru/articles/904876/> (дата обращения: 22.09.2025).
2. Quick start guide | PyCharm Documentation. — Текст : электронный // JetBrains s.r.o. : [сайт]. — URL: <https://www.jetbrains.com/help/pycharm/quick-start-guide.html> (дата обращения: 22.09.2025).