Программирование на языке Python

Среды разработки PyCharm и Jupiter Notebook



Горячая линия–Телеком

А. А. Бровкин, Н. А. Лаптев, А. Н. Пылькин,
Ю. С. Соколова, С. С. Тороян

Программирование на языке Python

Среды разработки PyCharm и Jupyter Notebook

Рекомендовано Научно-методическим советом Рязанского государственного радиотехнического университета им. В.Ф. Уткина для студентов вузов, обучающихся по укрупненному направлению подготовки 09.03.00 – «Информатика и вычислительная техника» (квалификация «бакалавр»)

Москва
Горячая линия – Телеком
2023

УДК 004.43(075.8)
ББК 32.973.2
П78

Р е ц е н з е н т :
профессор МГТУ им. Н. Э. Баумана, доктор техн. наук *В. А. Минаев*

А в т о р ы :
А. А. Бровкин, Н. А. Лаптев, А. Н. Пылькин,
Ю. С. Соколова, С. С. Тороян

П78 **Программирование** на языке Python. Среды разработки PyCharm и Jupyter Notebook: Учебное пособие для вузов / А. А. Бровкин, Н. А. Лаптев, А. Н. Пылькин и др. – М.: Горячая линия – Телеком, 2023. – 144 с.: ил.

ISBN 978-5-9912-1001-0.

Рассмотрены основные практические навыки работы в среде PyCharm. В качестве базовых элементов работы в среде PyCharm даны рекомендации по установке среды PyCharm, началу работы, правилам выполнения программы, дополнительным вариантам выполнения команд, переименованию переменных и базовой настройке IDE PyCharm.

Приведено описание и рассмотрены основные правила использования среды разработки Jupyter Notebook, которая, в первую очередь, ориентирована на программирование средствами Python. Среда Jupyter Notebook позволяет сразу видеть результаты выполнения программного кода, а также обеспечивает работу с отдельными его фрагментами. Среда разработки позволяет отдельно загрузить файл в память, отдельно проверить его содержимое и отдельно его обработать.

Для студентов вузов, изучающих дисциплину «Алгоритмические языки и программирование» укрупненного направления подготовки 09.03.00 – «Информатика и вычислительная техника», может быть использована в качестве практического руководства при изучении основ современных информационных технологий.

ББК 32.973.2

Адрес издательства в Интернет WWW.TECHBOOK.RU

Тиражирование книги начато в 2022 г.

Все права защищены.
Любая часть этого издания не может быть воспроизведена
в какой бы то ни было форме и какими бы то ни было средствами
без письменного разрешения правообладателя.

© ООО «Научно-техническое издательство «Горячая линия – Телеком»
www.techbook.ru
© А. А. Бровкин, Н. А. Лаптев,
А. Н. Пылькин, Ю. С. Соколова, С. С. Тороян

Предисловие

Язык программирования Питон (Python) на современном этапе развития средств разработки программных приложений является одним из массовых и популярным среди сообщества IT-специалистов. Широкое использование Python в учебном процессе в высших, средних специальных заведениях, а также в школах требует специального методического обеспечения учебного процесса, так как информация по методам программирования на Python носит, как правило, разрозненный и методически непродуманный характер (хотя подобной информации существует огромное множество). На начальных этапах освоения средств языка Python перед обучающимися остро встает вопрос выбора среды программирования, на основе которой осуществляется интегрированная разработка приложений на языке Python. Наиболее часто выбор приводит к интегрированной среде разработки (Integrated development environment, сокращенно — IDE) под названием PyCharm, которая является кросс-платформенной средой разработки с реализаций в Windows, MacOS и Linux. Среда PyCharm представляет собой достаточно сложную программу, которая обеспечивает выполнение множества действий по разработке программного обеспечения (ПО). Среда PyCharm интегрирует утилиты (вспомогательные системные программы):

- текстовый редактор;
- транслятор (компилятор и интерпретатор);
- отладчик;
- средства автоматизации сборки.

Опыт преподавания основ программирования показывает, что на первоначальных этапах изучения и освоения технологий разработки прикладного ПО обучающийся сталкивается с некоторыми сложностями работы в среде программирования, в частности в среде PyCharm. Эти сложности обусловлены низким уровнем знания английского языка, недостаточными навыками работы с компьютером и современным программным обеспечением. Для преодоления подобных трудностей на начальных ста-

диях освоение среды PyCharm и предназначено настоящее руководство для установки и эффективной работы со средствами IDE, ориентированными на разработку программ на языке Python. С этой точки зрения среда PyCharm содержит все необходимые средства для профессиональной разработки прикладного ПО. Использование PyCharm в процессе разработки программы позволяет обеспечить *высокую продуктивность*, обусловленную освобождением от рутинных операций и сосредоточением на выполнении более важных этапов, что приводит к экономии времени, а также *повышение качества кода*, что позволяет писать красивый программный код с одновременным контролем его качества путем проверки соответствия требованиям PEP8 — руководства по написанию кода на языке Python. *Помощь при написании кода* обеспечивается встроенным механизмом анализа программного кода, поиском ошибок и их быстрым исправлением, удобной навигацией по коду и другими функциями.

Для повышения эффективности разработки программ на языке Python часто используют другую среду программирования — Jupyter Notebook. Это весьма распространенное кроссплатформенное приложение является весьма мощным инструментом для интерактивной разработки программного кода, представления данных и их анализа. Веб-приложение запускается в браузере и обеспечивает выполнение следующих функций:

- редактирование и запуск кода на языке Python в браузере с подсветкой синтаксиса, автоотступами и автодополнением кода;
- отображение результатов вычислений с медиапредставлением (схемы, графики);
- работа с языком разметки Markdown и LATEX.

Введение

В данном учебном пособии рассмотрены основные правила работы со средой разработки PyCharm, которые необходимы для выполнения цикла практических работ по программированию на языке Python. Прежде чем перейти непосредственно к описанию теоретических сведений и практическим рекомендациям по выполнению практических работ, необходимо уточнить некоторые детали.

Предполагается, что цикл практических работ реализован с использованием алгоритмического языка Python версии 3.5.2, которая на момент постановки практических работ являлась одной их последних версий языка программирования. При этом следует иметь в виду, что работа с другой версией языка Python сопряжена с возможностью появления некоторых различий, что свойственно для различных версий языка Python.

В цикле практических работ использована свободно распространяемая интегрированная среда разработки IDE (Integrated Development Enviromnent) PyCharm Community Edition. Скачать ее можно по адресу

htpp://www.jetbrains.com/pycharm/download

Среда доступна для Windows, Linux, MacOS. Однако, если вам не нравится IDE, выбранная авторами, вы легко можете выбрать для себя другую среду разработки. Методика работы в различных средах отличается мало, более важным является соответствие версии интерпретатора Python.

В состав большинства IDE по умолчанию не входит интерпретатор языка Python. Его необходимо скачать и установить заранее. Сделать это можно с официального сайта

https://www.python.org/downloads/

Практические работы ориентированы на использование среды PyCharm, запущенной в ОС Windows. Для Linux и MacOS возможны свои особенности и отличия.

Рассмотренный материал не охватывает всего разнообразия возможностей, предоставляемых языком Python, как и не описывает всех методов эффективного написания программ на этом языке. За дополнительной информацией вы можете обратиться к книгам самоучителям, в том числе к тем, которые упоминаются в методических указаниях. Также крайне необходимо умение правильно искать информацию с помощью поисковых систем в Интернете, что помогает оперативно решать возникающие трудности.

В процессе выполнения практических работ, как правило, требуется делать отчеты по проделанной работе. В отчет обычно входит схема алгоритма. Для удобного создания схемы в электронном виде с возможностью ее дальнейшего экспорта в векторный или растровый формат изображения вы можете воспользоваться бесплатным сервисом www.draw.io в виде онлайн-редактора на сайте или офлайн-надстройки для браузера (тестировалось для Google Chrome).

Во второй части учебного пособия рассмотрены основные правила работы в среде программирования Jupyter Notebook. Приведены сведения по следующим основным вопросам:

- установка среды Jupyter Notebook;
- терминология и основные определения;
- интерфейс и правила работы;
- интерактивное управление с использованием видиктов (графических элементов — кнопок, выпадающих списков, текстовых полей).

1 Установка и запуск среды PyCharm

Язык Python («пайтон»; в русском языке распространено название Питон) — высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен. В то же время стандартная библиотека включает большой объём полезных функций.

Python поддерживает несколько парадигм программирования, в том числе структурное, объектно-ориентированное, функциональное, императивное и аспектно-ориентированное. Основные архитектурные черты: динамическая типизация, автоматическое управление памятью, полная интроспекция, механизм обработки исключений, поддержка многопоточных вычислений и удобные высокоуровневые структуры данных. Код в Python организовывается в функции и классы, которые могут объединяться в модули (они в свою очередь могут быть объединены в пакеты).

Установка среды PyCharm

В том случае, если вы хотите установить среду PyCharm на своем ПК, вам необходимо:

1) скачать по ссылке интерпретатор языка Python (желательно версии 3.52, чтобы не возникало расхождений с данными методическими указаниями) и установить его. Замечание: в первом окне мастера установки программы поставьте галочку на опции «Add Python 3.5 to PATH», иначе в дальнейшем PyCharm не увидит установленного интерпретатора;

2) скачать среду PyCharm (бесплатную версию Community Edition) и установить ее по ссылке

http://www.jetbrains.com/pycharm/download

3) запустить PyCharm и пройти начальную настройку (выбор цветовой схемы оформления и т. п.; можно просто нажать несколько раз «далее» и настроить эти параметры позже через меню настроек среды).

Начало работы

Запустить PyCharm можно через ярлык на рабочем столе или из меню Пуск: Все программы → JetBrains → JetBrains PyCharm Community Edition. При этом вы увидите начальное окно работы со средой (рис. 1.1). Здесь вам будет предложено создать новый проект или выбрать ранее созданный. В списке в левой части окна представлены последние открытые проекты.

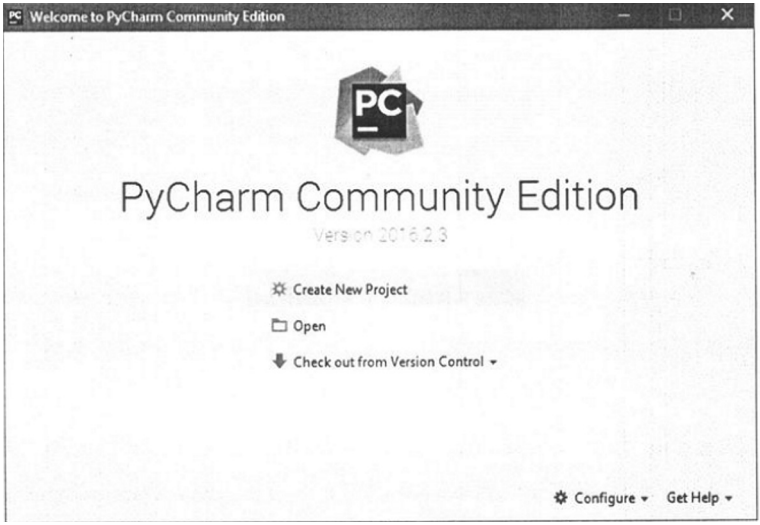


Рис. 1.1. Стартовое окно среды PyCharm

Нажмите на надпись Create New Project, чтобы создать ваш первый проект, Вы увидите окно, в котором вам предложено будет выбрать папку для хранения проекта и интерпретатор (в списке должен быть хотя бы один интерпретатор, установленный ранее). Укажите путь к папке и ее имя.

Необходимо заметить, что в одном проекте можно хранить несколько файлов с программами, т. е. вы можете создать всего один проект и выполнять в нем все свои лабораторные работы, создавая для каждой новой работы отдельный файл кода Python в корне проекта. Это позволит вам оперативно сверяться с уже выполненными заданиями и найденными ранее решениями проблем.

Если вы создали новый рабочий проект, то увидите окно, аналогичное представленному на рис. 1.2.

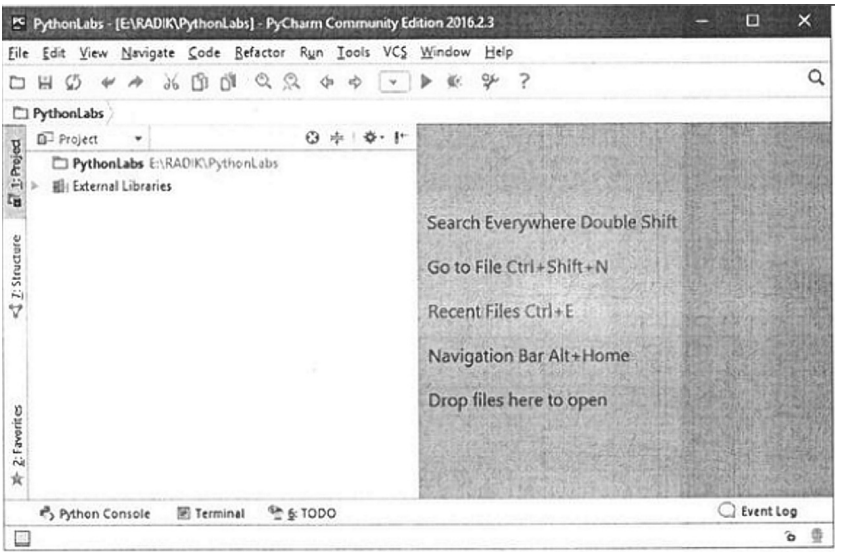


Рис. 1.2. Рабочее представление среды

На данный момент проект пустой и в него необходимо добавить первый файл Python. Для этого нажмите правую кнопку мыши на имени вашего проекта в обозревателе решений (в левой части окна) и выберите в контекстном меню команду New → Python File (рис. 1.3). В появившемся окне укажите имя файла, например «Лабораторная 1».

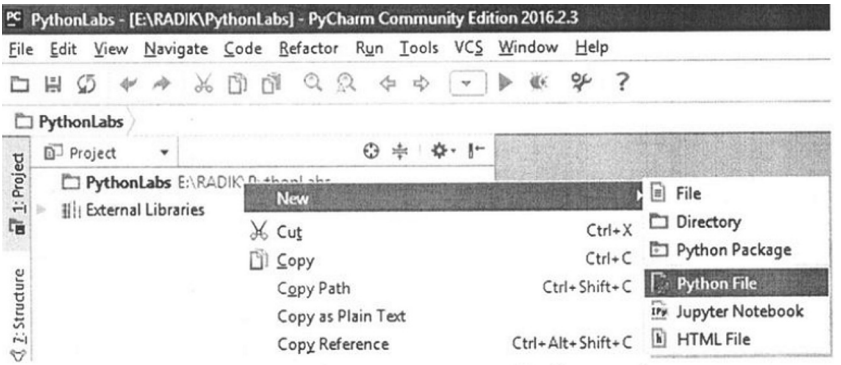


Рис. 1.3. Создание нового файла кода

После создания нового файла Python он автоматически будет открыт на редактирование в основной части окна. На данный

момент он пустой. Первая лабораторная работа подразумевает копирование готового текста программы из пособия и взаимодействие с ним.

Выполнение программы

После того как вы написали программный код в созданном ранее файле Python, его необходимо запустить. Существует два режима запуска программы: обычный (Run) и отладочный (Debug). С особенностями режима отладки вы ознакомитесь позже при выполнении следующих практических работ. Для запуска программы в обычном режиме необходимо нажать сочетание клавиш Alt+Shift+F10. При этом появится окно с выбором программы для запуска. Выберите свою программу и нажмите клавишу Enter. (рис. 1.4). Повторный запуск программы (например, после внесения изменений) можно быстро осуществить нажатием комбинации клавиш Shift+F10 или экранной кнопки в виде зеленой стрелки в верхней части окна среды.

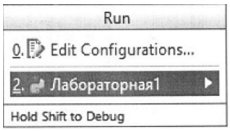


Рис. 1.4. Запуск программы на выполнение

После запуска программы вы увидите терминальное окно с результатом выполнения вашей программы. (рис. 1.5). В случае, если в программе присутствует ввод значений с клавиатуры (оператор input()), его необходимо осуществлять именно в этом окне.



Рис. 1.5. Терминал ввода-вывода

Вы можете управлять процессом выполнения программы с помощью экранных кнопок в левой части терминального окна. В частности, можно запустить программу заново, поставить выполнение программы на паузу или остановить ее выполнение полностью. Также там есть кнопки прокрутки в конец консоли, очистки консоли и печати содержимого консоли. Однако печать результатов напрямую из консоли для отчета по лабораторной работе не рекомендуется, так как намного лучше скопировать результаты выполнения в основной документ отчета и отформатировать их должным образом. (Напоминаем, что тест программы обязательно должен быть представлен моноширинным шрифтом, например Consolas или Courier New. Результаты выполнения программы также желательно печатать моноширинным шрифтом, в котором все знаки имеют одинаковую ширину).

Дополнительные варианты выполнения команд

Как видно из рис. 1.5, в нижней части терминального окна есть вкладки. Одна из них активна при запуске программы в обычном режиме — собственно, вкладка Run. Также при выполнении программы в отладочном режиме там появится аналогичная вкладка Debug. Однако в данный момент нам интересны две другие вкладки, которые уже отображены рядом с активной. Это Python Console и Terminal. Вкладка Terminal — это окно, представляющее собой обычный терминал ОС Windows, запущенный из папки вашего проекта. Если вы в достаточной мере владеете командами DOS, эта вкладка может оказаться крайне полезной. Вкладка Python Terminal заслуживает отдельного внимания. Дело в том, что программы на языке Python могут выполняться двумя способами: целым файлом и построчно. Когда вы запускаете файл на выполнение, вы используете первый способ. А во вкладке Python Terminal вы можете сделать это вторым способом — вводя команды последовательно по частям. При этом все введенные переменные запоминаются: если вы введете «x=42», а затем «print(x)», то программа вернет вам введенное значение 42. Также этот терминал можно использовать как калькулятор: введя команду «2*3», вы получите результат «= 6».

Переименование переменных

Часто бывает необходимо изменить имя переменной (например, на более информативное). При этом, если программа достаточно большая и переменная входит в текст программы много

раз и в различных местах, то менять ее имя вручную крайне сложно. Более того, возрастает вероятность ошибки. Однако в PyCharm есть встроенная функция переименования переменных. Для этого выбелите имя переменной в любом месте программы и нажмите комбинацию клавиш Shift+F6. В появившемся окне введите новое имя переменной и нажмите кнопку Refactor. Если имя вашей переменной также может встречаться в комментариях или строковых константах, не забудьте поставить галочку в окне переименования для опции Search in comments and strings.

Базовая настройка IDE PyCharm

Вы можете гибко настроить данную среду под себя. В PyCharm присутствует множество настроек. Вызвать меню настроек можно по пути File → Settings или нажатием клавиш Ctrl+Alt+S. Однако большинство настроек уже выбраны оптимально. Самое главное, что вы можете смело настроить сразу внешний вид среды. Здесь есть две цветовых схемы — для всего интерфейса (Appearance Behavior → Appeampce → Theme) и отдельно для **области** редактора кода (Editor → Colors and Fonts → Scheme). Вы также можете создать свою цветовую схему для редактора кода, скопировав одну из системных схем нажатием по кнопке Save As и изменив нужные параметры в подпунктах меню Colors Fonts. Также вы можете настроить комбинации горячих клавиш (или выбрать одну из готовых схем) в меню настроек во вкладке Keymap. Все горячие клавиши в данном пособии указаны для схемы Default.

Пример программы

```
\\#!/usr/bin/env python
# -*- coding:  utf-8 -*-
import math
def p1():
  x = 1.0
  while x < 10.0:
    print x, "\t", math.log(x)
    x += 1.0
def main():
  p1()
  return 0
if __name__ == '__main__':
  main()
```

Строка "\t" обозначает знак табуляции. Благодаря ему значения выстраиваются в два столбца. Разберем, как эта программа работает. Параметр x изменяется от 1.0 с шагом 1.0, пока он меньше 10.0. В теле цикла выводится текущее значение параметра x, затем знак табуляции и результат вычисления функции math.log(x), т. е. натуральный логарифм от x.

Результат работы программы будет выглядеть так:

```
1.0  0.0
2.0  0.69314718056
3.0  1.09861228867
4.0  1.38629436112
5.0  1.60943791243
6.0  1.79175946923
7.0  1.94591014906
8.0  2.07944154168
9.0  2.19722457734
```

2 Инструкция по началу работы в PyCharm

На каких языках поддерживается PyCharm?

С PyCharm вы можете разрабатывать приложения на языке Python. Кроме того, в Professional Edition можно разрабатывать приложения Django, Flask и Pyramid. Кроме того, он полностью поддерживает HTML (включая HTML5), CSS, JavaScript и XML: эти языки включены в IDE через плагины и включены для вас по умолчанию. Поддержка других языков и фреймворков также может быть добавлена через плагины (перейдите к **Settings | Plugins** или **PyCharm | Preferences | Plugins** для пользователей MacOS, чтобы узнать больше или установить их во время первого запуска IDE).

На каких платформах я могу запустить PyCharm?

PyCharm — это кроссплатформенная среда разработки, работающая в Windows, MacOS и Linux. Если вам нужна помощь в установке PyCharm, см. Инструкции по установке для Linux, macOS и Windows.

Шаг 1. Откройте/создайте проект в PyCharm

Зачем мне нужен проект?

Все, что вы делаете в PyCharm, выполняется в контексте проекта. Он служит основой для поддержки кодирования, рефакторинга, согласованности стиля кодирования и т. д. У вас есть два варианта начать работу над проектом внутри среды IDE.

1. Открыть существующий проект

В начале откройте один из ваших существующих проектов, хранящихся на вашем компьютере. Вы можете сделать, нажав «Открыть проект» (**Open**) на экране приветствия (рис. 2.1).

Или выберите Open в меню File и укажите каталог, в котором находятся ваши источники. Затем PyCharm создаст для вас



Рис. 2.1. Экран приветствия

проект из ваших источников. Смотрите также раздел «Импорт проекта» из существующего исходного кода.

2. Создать проект с нуля.

Если вы предпочитаете начинать с нуля, нажмите New Project, на открывшемся экране введите имя проекта в диалоговом окне и будет создан проект Python.

Шаг 2. Ознакомьтесь с интерфейсом

Когда вы запускаете PyCharm в первый раз или когда нет открытых проектов, вы видите Экран приветствия. Он предлагает вам основные точки входа в среду IDE: создание или открытие проекта, проверка проекта с помощью контроля версий, просмотр документации и настройка среды IDE. Когда проект открывается, вы видите главное окно, разделенное на несколько логических областей. Рассмотрим ключевые элементы пользовательского интерфейса на рис. 2.2.

1. **Project Tool Window.** *Панель инструментов проекта.* В этом окне отображаются файлы вашего проекта.

2. **PyCharm Editor.** *Редактор PyCharm.* Находится с правой стороны, где вы пишете свой код. В нем есть вкладки для удобной навигации между открытыми файлами.

3. **Navigation Bar.** *Панель навигации.* Находится над редактором, позволяет быстро запускать и отлаживать ваше приложение, а также выполнять процедуры контроля версий VCS.

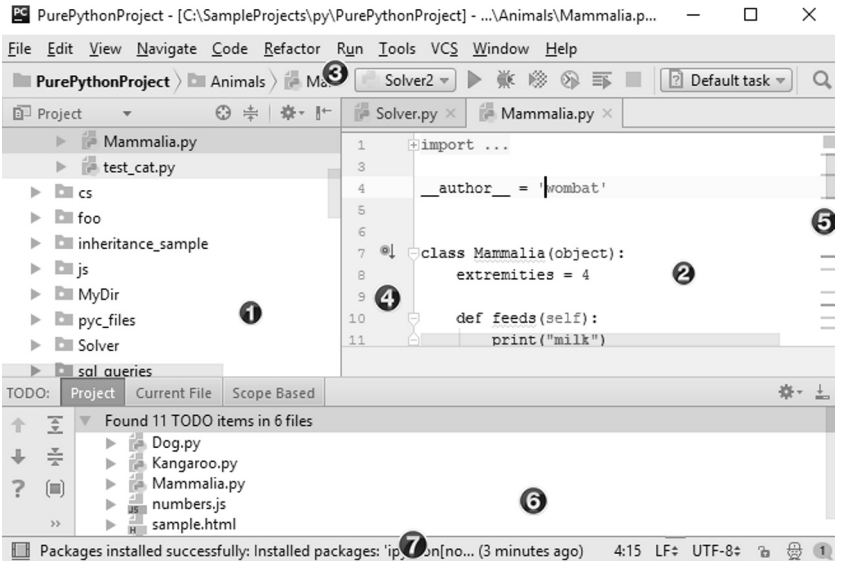


Рис. 2.2. Окно среды

4. **Left gutter.** Левый столбец, вертикальная полоса рядом с редактором. Показывает брекпойнты и обеспечивает удобный способ перехода по иерархии кода. Он также отображает номера строк и историю VCS.

5. **Right gutter.** Правый столбец, справа от редактора. PyCharm постоянно контролирует качество вашего кода и постоянно показывает результаты проверки в правом столбце: ошибки, предупреждения и т. д. Индикатор в правом верхнем углу показывает общий статус проверки кода для всего файла.

6. **PyCharm Tool Windows**. *Панель инструментов PyCharm.* Это специальные окна, прикрепленные к низу и сторонам рабочей области, которые обеспечивают доступ к типичным задачам, таким как управление проектами, поиск и навигация по исходному коду, интеграция с системами контроля версий и т. д.

7. **Status Bar.** *Строка состояния.* Указывает состояние вашего проекта и показывает различные предупреждения и информационные сообщения.

Шаг 3. Настройте свою среду

Вы можете настроить среду IDE, чтобы она идеально соответствовала вашим потребностям и была удобна для Вас. Зайдите в меню **File/Settings**, чтобы просмотреть список доступных параметров настройки (рис. 2.3 и 2.4).

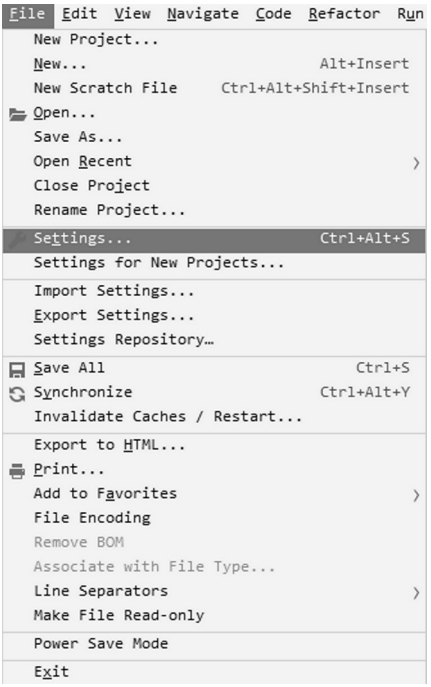


Рис. 2.3. Окно File

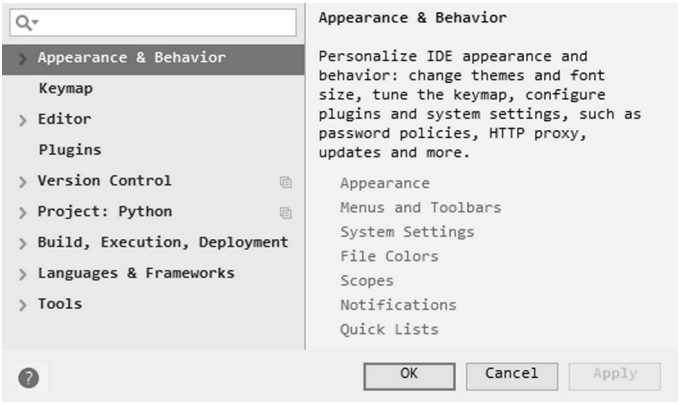


Рис. 2.4. Окно Settings

Шаг 4. Сделайте свой код аккуратным

PyCharm контролирует ваш код и пытается сохранить его точным и чистым. Он обнаруживает потенциальные ошибки и проблемы и предлагает быстрые исправления (quick-fixes) для

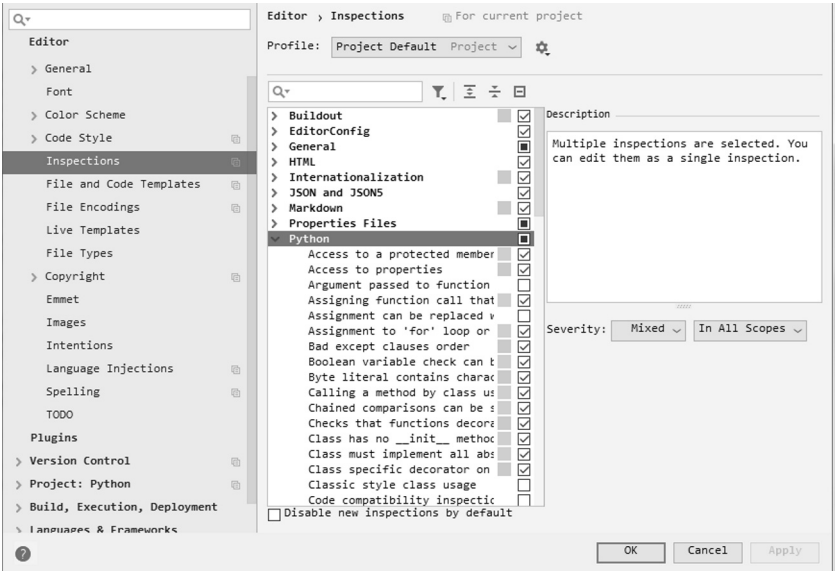


Рис. 2.5. Окно Settings

них. Каждый раз, когда среда IDE находит неиспользованный код, бесконечный цикл и многое другое, что, вероятно, потребует вашего внимания, вы увидите лампочку. Щелкните по ней или нажмите Alt+Enter, чтобы применить исправление. Полный список доступных проверок можно найти в разделе меню **File/Settings/Editor/Inspections** (рис. 2.5). Отключите некоторые из них или включите другие, а также настройте степень серьезности каждой проверки. Вы решаете, следует ли считать это ошибкой или просто предупреждением.

Шаг 5. Создание кода

Написание кода может быть намного проще и быстрее, если вы используете варианты генерации кода, доступные в PyCharm. **The Code/Generate** (Alt+Insert) поможет вам создать символы, а также предложит переопределить/реализовать некоторые функции.

3 Перевод и назначение инструментов среды PyCharm

Information

Вы всегда можете обратиться к англоязычной справке по среде PyCharm нажатием клавиши F1. Русскоязычной справки (как и русифицированной версии среды) на данный момент не существует.

Окно среды

При входе в программу PyCharm пользователя будет встречать окно (рис. 3.1).

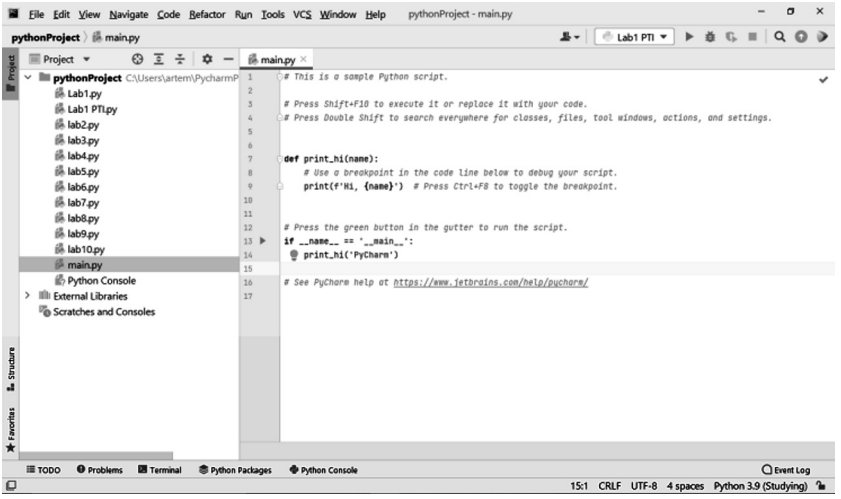


Рис. 3.1. Окно среды

Рассмотрим подробнее окно среды.

В левой части экрана расположена часть управления проектом (рис. 3.2).

В табл. 3.1 представлены перевод и назначение основных функций управления проектом.

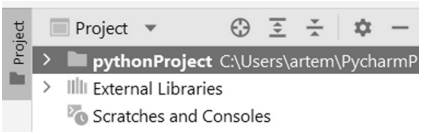


Рис. 3.2. Управление проектом

Таблица 3.1

Функции управления проектом

Название на английском	Перевод на русский	Назначение
Project	Проект	Просмотр проекта
External libraries	Внешние библиотеки	Просмотр внешних библиотек
Select opened file	Выбрать открытый (текущий) файл	
Expand all	Раскрыть всё	
Collapse all	Свернуть всё	
Show options menu	Показать меню настроек	Для настройки удобного обращения с проектом
Show members	Показывать элементы	
Show excluded files	Показывать исключенные файлы	
Enable preview tab	Включить вкладку предварительного просмотра	
Open files with single click	Открывать файлы с одного щелчка мыши	
Always select opened file	Всегда выбирать открытый файл	
Sort by type	Сортировать по типу	
Folders always on top	Папки всегда сверху	
Edit scopes	Области редактирования	
File nesting	Вложенные файлы	
Group tabs	Группировать вкладки	
View mode	Режим отображения	
Move to	Перейти к...	Для перемещения вкладки в какую-либо часть среды программирования
Resize	Изменить размер	
Remove from sidebar	Удалить с боковой панели	
Help	Руководство	

Панель проверки

В нижней части среды программирования PyCharm имеются панели (рис. 3.3).



Рис. 3.3. Функции проверки

Панель Problems

Панель предназначена для выявления неисправностей. Панель Problems представлена на рис. 3.4.

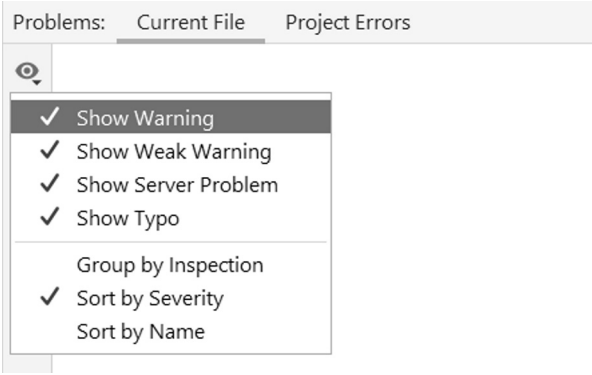


Рис. 3.4. Панель Problems

В табл. 3.2 показаны перевод и назначение основных функций панели Problems.

Таблица 3.2

Функции панели Problems

Название на английском	Перевод на русский	Назначение
Current file	Текущий файл	
Project errors	Ошибки в проекте	
Show warning	Показывать предупреждения	
Show weak warning	Показывать незначительные предупреждения	
Show server problem	Показывать проблемы с сервером	
Show typo	Показывать опечатки	
Group by inspection	Группировать в порядке проверки	Группирует ошибки по порядку их проверки

Окончание табл. 3.2

Название на английском	Перевод на русский	Назначение
Sort by severity	Группировать по тяжести ошибки	
Sort by name	Группировать по названию (имени)	Группировка ошибок в алфавитном порядке

Панель Terminal

Панель Terminal (Терминал) представлена на рис. 3.5.

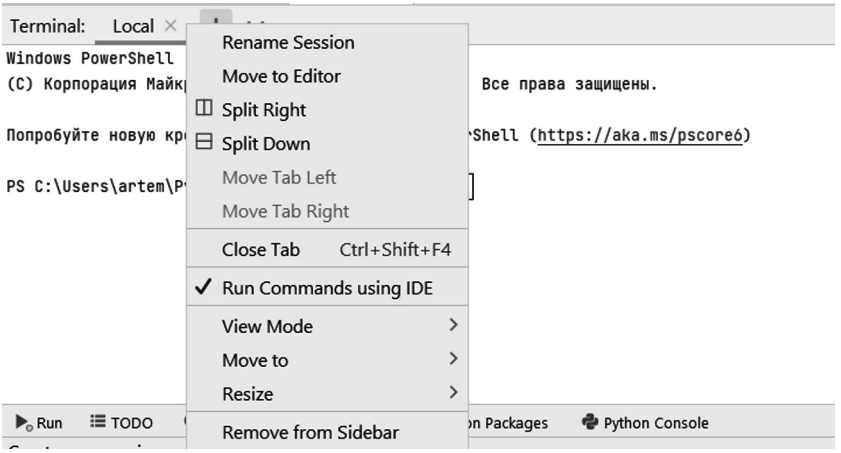


Рис. 3.5. Панель Terminal

В табл. 3.3 показаны перевод и назначение основных функций панели Terminal.

Таблица 3.3

Функции панели Terminal

Название на английском	Перевод на русский	Назначение
Local	Название текущего сеанса	Можно переходить по разным сеансам
“+”	Добавить новый сеанс	
Rename session	Переименовать сеанс	
Move to editor	Перейти в редактор	
Split right	Разделить справа	Создать еще один терминал текущего сеанса справа
Split down	Разделить снизу	Создать еще один терминал текущего сеанса снизу

Окончание табл. 3.3

Название на английском	Перевод на русский	Назначение
Close tab	Закрыть вкладку	
Run commands using IDE	Запускать команды, используя интегрированную среду разработки	
View mode	Режим отображения	
Move to	Перейти к...	Для перемещения вкладки в какую-либо часть среды программирования
Resize	Изменить размер	
Remove from sidebar	Удалить с боковой панели	

Панель Python Packages

Панель Пакеты Python нужна для просмотра документации пакетов. Панель Python Packages представлена на рис. 3.6.

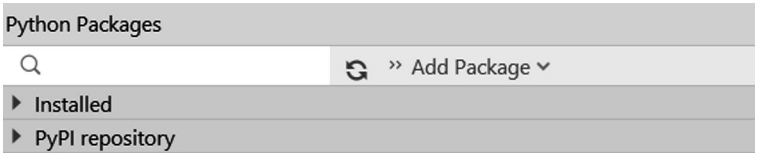


Рис. 3.6. Панель Python Packages

В табл. 3.4 показаны перевод и назначение основных функций панели Python Packages.

Таблица 3.4

Функции панели Python Packages

Название на английском	Перевод на русский	Назначение
Installed	Установленные	Список установленных пакетов
PyPI repository	Репозиторий PyPI	Пакеты, которые можно установить
Add package	Добавить пакет	

Панель Python Console

Панель предназначена для того, чтобы проверять работу отдельных строк кода.

Панель Python Console представлена на рис. 3.7.

В табл. 3.5 показаны перевод и назначение основных функций панели Python Console.

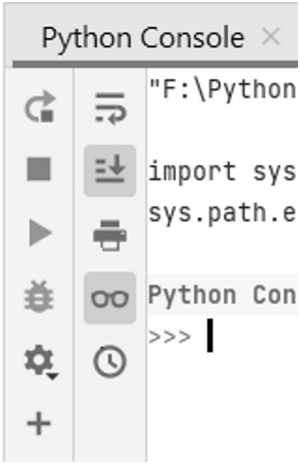


Рис. 3.7. Панель Python Console

Таблица 3.5

Функции панели Python Console

Кнопка	Назначение
	Перезапустить код
	Остановить консоль
	Выполнить текущую инструкцию в однолинейной консоли
	Подключить отладчик
	Настройки консоли
	Создать новое окно консоли
	Отобразить всё в зоне видимости
	Переместиться в конец
	Распечатать...
	Показать переменные
	Просмотреть историю запросов

Верхняя правая часть окна среды

Панель предназначена для того, чтобы проверять работу отдельных строк кода.

Верхняя правая часть окна среды представлена на рис. 3.8.

Таблица 3.6

Функции верхней правой части окна среды

Кнопка	Назначение
	Выбрать конфигурацию для запуска или отладки
	Выполнить код
	Выполнить отладку
	Выполнить код с применением защиты
	Остановить
	Поиск по всему в среде программирования
	Обновления



Рис. 3.8. Верхняя правая часть окна среды

В табл. 3.6 показаны перевод и назначение основных функций верхней правой части окна среды.

Верхняя панель

Верхняя панель представлена на рис. 3.9.



Рис. 3.9. Исходная верхняя панель

Меню File

Меню File (Файл) представлено на рис. 3.10.

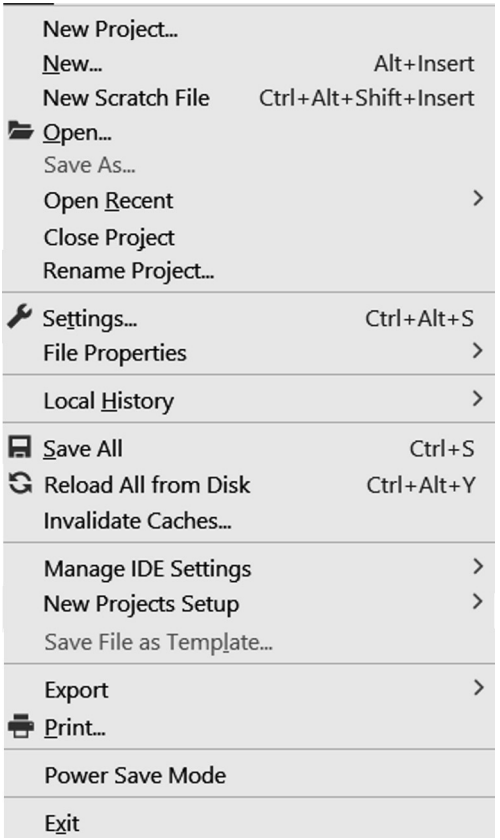


Рис. 3.10. Меню File

В табл. 3.7 показаны перевод и назначение основных функций меню File.

Таблица 3.7

Функции меню *File*

Название на английском	Перевод на русский	Назначение
New Project	Создать новый проект	Для создания нового проекта
New	Создать новый файл	Для создания нового элемента в проекте
New Scratch File	Создать черновой файл	
Open	Открыть файл	Для открытия указанного каталога или существующего проекта PyCharm
Save As	Сохранить как	Для сохранения файла, открытого в данный момент в редакторе, в указанном каталоге
Open Recent	Открыть недавние	
Close Project	Закрыть проект	
Rename Project	Переименовать проект	
Settings	Настройки	
File properties	Настройки файла	
Local History	История изменений	
Save All	Сохранить всё	Для сохранения всех изменений после редактирования
Reload All from Disk	Перезагрузить всё с диска	
Invalidate Caches	Очистка кэша проекта	
Manage IDE Settings	Настройки IDE	
New Projects Setup	Настройки новых проектов	
Save File as Template...	Сохранить файл как шаблон...	
Export	Экспорт в HTML	
Print	Печать	
Add to Favorites	Добавить в избранное	
Power Save Mode	Режим энергосбережения	
Exit	Выход из программы	Выход из PyCharm

Меню Edit

Меню Edit (Правка) представлено на рис. 3.11.

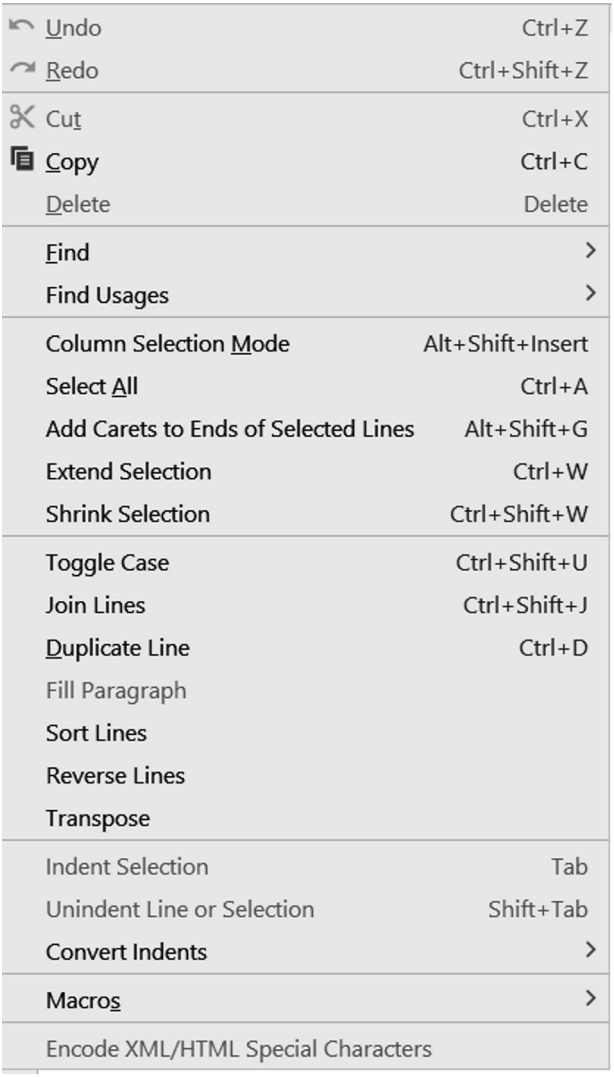


Рис. 3.11. Меню *Edit*

В табл. 3.8 показаны перевод и назначение основных функций меню Edit.

Таблица 3.8

Функции меню *Edit*

Название на английском	Перевод на русский	Назначение
Undo	Отменить последнее действие	
Redo	Вернуть отмененное	
Cut	Вырезать	
Copy	Копировать	
Copy Path/ Reference	Скопировать информацию о файле	
Paste	Вставить	
Delete	Удалить	
Find	Найти	Для открытия подменю команд поиска
Find Usages	Варианты использования	
Column Selection Mode	Режим «выделения столбца»	
Select All	Выделить всё	
Add Carets to Ends of Selected Lines	Установить «каретку» для выделенной строки	
Extend Selection	Расширенное выделение	
Shrink Selection	Сокращенное выделение	
Toggle Case	Изменить регистр слова	
Join Lines	Соединить строки	Для объединения строк или литералов
Duplicate Line	Дублировать строки	
Fill Paragraph	Заполнить абзац	
Sort Lines	Перестроить программу	
Reverse Lines	Перевернуть программу	
Transpose	Транспонировать символы	
Duplicate Line	Дублировать строки	
Fill Paragraph	Заполнить абзац	
Sort Lines	Перестроить программу	
Reverse Lines	Перевернуть программу	
Transpose	Транспонировать символы	
Indent Selection	Отступ вправо	
Unindent Line or Selection	Отступ влево	
Convert Indents	Изменить отступ на...	
Macros	Макросы	Для открытия подменю макросов
Encode XML/ HTML Special Characters	Кодирование специальных символов XML/HTML	

Меню View

Меню View (Просмотр) представлено на рис. 3.12.

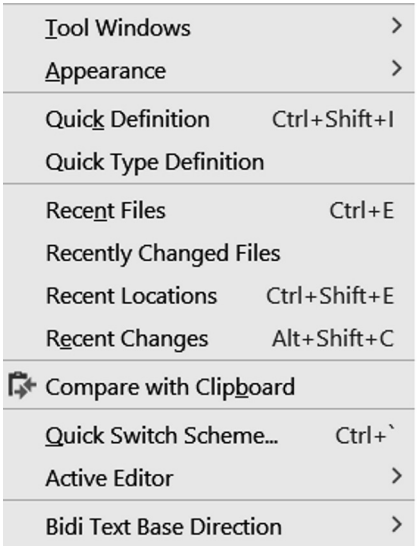


Рис. 3.12. Меню *View*

В табл. 3.9 показаны перевод и назначение основных функций меню View.

Таблица 3.9

Функции меню *View*

Название на английском	Перевод на русский	Назначение
Tool Windows	Инструменты	Для открытия списка доступных окон инструментов
Appearance	Внешний вид	
Quick Definition	Кратное описание элемента	
Quick Type Definition	Определение типа элемента	
Quick Documentation	Информация о выбранном элементе	
Parameter Info	Информация о параметре	
Type Info	Информация о типе	
Context Info	Справка о...	
Error Description	Описание ошибки	
Recent Files	Последние файлы	Для открытия списка последних изменений

Окончание табл. 3.9

Название на английском	Перевод на русский	Назначение
Recently Changed Files	Недавно измененные файлы	
Recent Locations	Недавние перемещения по коду	
Recent Changes	Недавние изменения	
Compare With...	Сравнить с...	
Compare with Clipboard	Сравнение с буфером обмена	
Quick Switch Scheme	Сменить цветовую тему	
Active Editor	Меню редактора	Для открытия списка вложенных команд проверки
Bidi Text Base Direction	Базовое направление текста	

Меню Navigate

Меню Navigate (Поиск, навигация) представлено на рис. 3.13.

В табл. 3.10 показаны перевод и назначение основных функций меню Navigate.

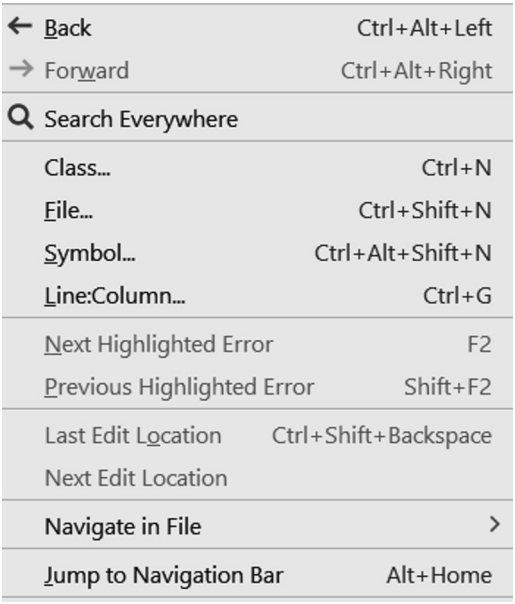


Рис. 3.13. Меню *Navigate* (начало)

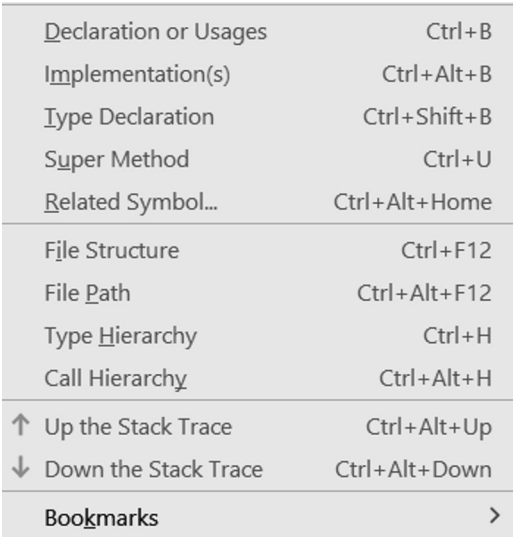


Рис. 3.13. Меню *Navigate* (окончание)

Таблица 3.10

Функции меню *Navigate*

Название на английском	Перевод на русский	Назначение
Back	Назад	
Forward	Вперед	
Search Everywhere	Искать везде	Для поиска элемента в проекте или за его пределами по его имени
Class...	Искать класс...	
File...	Искать файл...	
Symbol...	Искать символ...	
Line:Column...	Перейти к строке:столбцу	
Next Highlighted Error	Следующая выделенная ошибка	
Previous Highlighted Error	Предыдущая выделенная ошибка	
Last Edit Location	Место последнего редактирования	
Next Edit Location	Следующее место редактирования	
Navigate in File	Поиск в файле...	
Select In...	Выбрать...	

Окончание табл. 3.10

Название на английском	Перевод на русский	Назначение
Jump to Navigation Bar	Перейти к панели навигации	
Declaration or Usages	Перейти к объявлению переменной, класса, метода	
Implementation(s)	Перейти к реализации	
Type Declaration	Определить тип и перейти к его реализации	
Super Method	Функция Super()	
Related Symbol...	Связанный символ	
File Structure	Структура файла	
File Path	Путь к файлу	Для поиска пути к файлу
Type Hierarchy	Иерархия наследования текущего класса и переход по ней	
Call Hierarchy	Иерархия вызовов выбранного метода	
Up the Stack Trace	Вверх по трассировке стека	
Down the Stack Trace	Вниз по трассировке стека	
Bookmarks	Закладки	

Меню Code

Меню Code (Код) представлено на рис. 3.14.

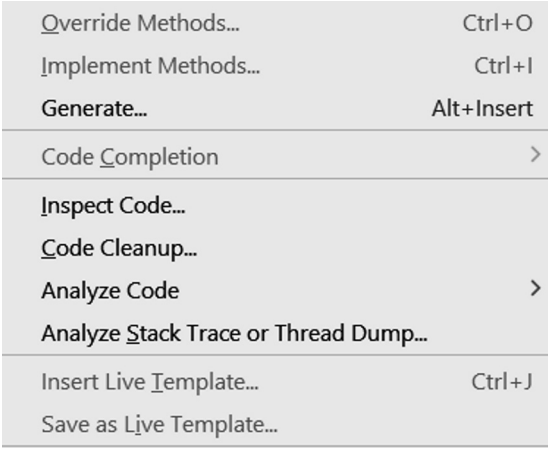


Рис. 3.14. Меню *Code* (начало)

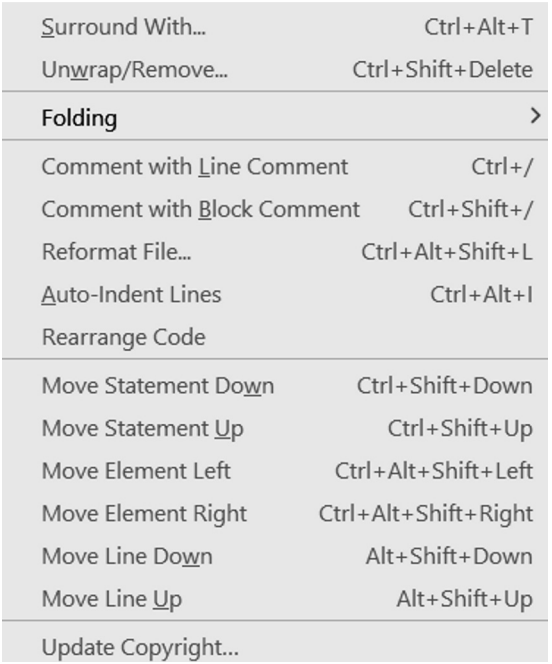


Рис. 3.14. Меню *Code* (окончание)

В табл. 3.11 показаны перевод и назначение основных функций меню Code.

Таблица 3.11

Функции меню *Code*

Название на английском	Перевод на русский	Назначение
Override Methods...	Переопределить метод родительского класса	Для переопределения метода суперкласса
Implement Methods...	Реализовать интерфейс	
Generate...	Генератор кода — сеттеров, зависимостей в pom.xml и т. д.	Для создания нового элемента
Code Completion	Автозавершение кода	
Inspect Code...	Проверить участок кода	
Code Cleanup...	Очистка участка кода	
Analyze Code	Анализ кода	
Analyze Stack Trace or Thread Dump...	Анализ трассировки стека	

Окончание табл. 3.11

Название на английском	Перевод на русский	Назначение
Insert Live Template...	Генерация шаблонного кода	
Save as Live Template...	Сохранить как шаблонный код	
Surround With...	Добавить к выделенному коду управляющую конструкцию	
Unwrap/Remove...	Вынуть код из управляющей конструкции	
Folding	Свернуть или развернуть фрагмент кода	Для открытия вложенных команд сворачивания
Comment with Line Comment	Закомментировать/раскомментировать текущую строку	
Comment with Block Comment	Закомментировать/раскомментировать выделенный код	
Reformat Code	Приведение кода в соответствие Code-Style	
Reformat File...	Приведение файла в соответствие Code-Style	
Auto-Indent Lines	Выравнивание отступов в коде	
Optimize Imports	Оптимизация импортов	
Rearrange Code	Перестроить код	
Move Statement Down	Опустить выделенное выражение	
Move Statement Up	Поднять выделенное выражение	
Move Element Left	Переместить элемент влево	
Move Element Right	Переместить элемент вправо	
Move Line Down	Опустить строку	
Move Line Up	Поднять строку	
Update Copyright...	Обновление авторских прав	

Меню Refactor

Меню Refactor (Перепроектирование кода) представлено на рис. 3.15.

В табл. 3.12 показаны перевод и назначение основных функций меню Refactor.

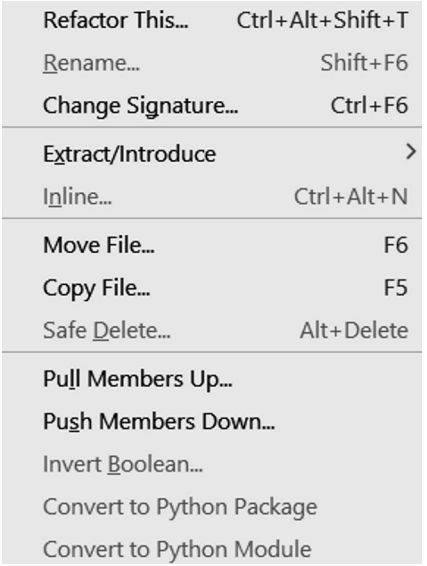


Рис. 3.15. Меню *Refactor*

Таблица 3.12

Функции меню *Refactor*

Название на английском	Перевод на русский	Назначение
Refactor This...	Рефакторинг (изменение структуры кода)	
Rename...	Переименование переменной, класса и т. д.во всем коде	Для переименования элемента
Change Signature...	Изменение сигнатуры метода во всем коде	
Extract/Introduce	Извлечь из кода/Ввести в код...	
Inline...	Инлайнинг метода, переменной, аргумента	Для выполнения встроенного рефакторинга
Move...	Создать копию класса, файла или каталога	
Copy File...	Безопасное удаление класса, метода или атрибута	
Safe Delete...		
Pull Members Up...	Переместить метод класса по иерархии вверх	

Окончание табл. 3.12

Название на английском	Перевод на русский	Назначение
Push Members Down...	Переместить метод класса по иерархии вниз	
Invert Boolean...	Инвертировать логическое значение...	
Convert to Python Package	Преобразование в контейнер	
Convert to Python Module	Преобразовать в модуль	

Меню Run

Меню Run представлено на рис. 3.16.

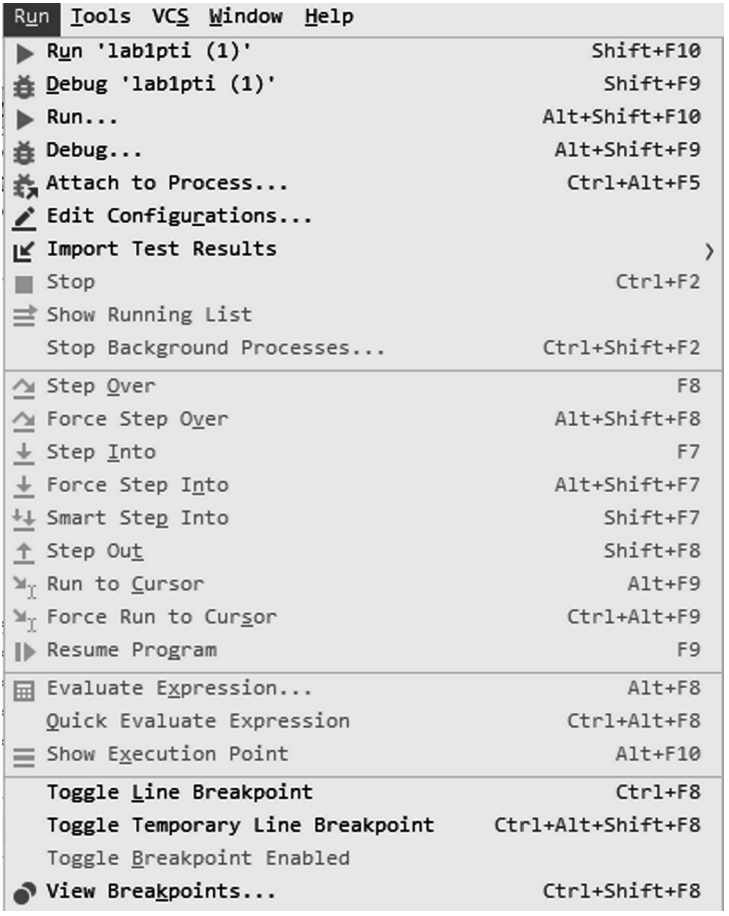


Рис. 3.16. Меню *Run*

В табл. 3.13 показаны перевод и назначение основных функций меню Run.

Таблица 3.13

Функции меню *Run*

Название на английском	Перевод на русский	Назначение
Run	Запустить	Запуск выбранного процесса
Debug	Отладка	Отладка выбранного процесса
Attach to Process	Присоединение к процессу	
Edit Configurations	Редактирование конфигураций	
Import Test Results	Импорт результатов тестирования	
Stop	Остановить выполнение	
Show Runnig List	Показ списка запущенных программ	Завершение текущего процесса
Stop Background Proscesses	Останов фоновых процессов	
Step Over	Перешагнуть	
Force Step Over	Принужденно перешагнуть	Переход к следующей строке в этом файле
Step Into	Входить, шаг в	
Force Step Into	Принудительно вмешаться	Переход к следующей выполненной строке, следующие вызовы функций
Smart Step Into	Разумный шаг в	Переход к следующей выполненной строке, даже если она находится в файле, из которого был исключен
Step Out	Выходить, шаг наружу	
Run To Cursor	Переход к курсору	Переход к следующей строке в коде, который вызвал эту функцию
Force Run To Cursor	Принудительный переход к курсору	Переход к точке, на которой установлен курсор
Resume Program	Программа резюме	
Evaluate Expression	Оценка выражения	
Quick Evaluate Expression	Быстрая оценка выражения	

Окончание табл. 3.13

Название на английском	Перевод на русский	Назначение
Show Execution Point	Показ точки выполнения	
Restore Breakpoint	Восстановление точки прерывания	
Toggle Line Breakpoint	Точка разрыва строки	
Toggle Temporary Line Breakpoint	Переключение временной точки разрыва строки	
Toggle Breakpoint Enabled	Точка разрыва строки включен	
View Breakpoints	Просмотр точек разрыва	

Меню Tools

Меню Tools (Инструменты) представлено на рис. 3.17.

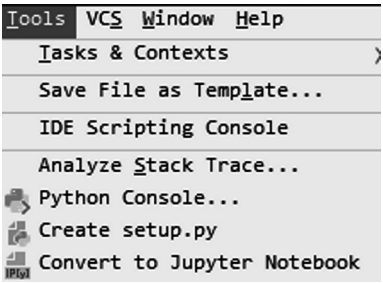


Рис. 3.17. Меню *Tools*

В табл. 3.14 показаны перевод и назначение основных функций меню Tools.

Таблица 3.14

Функции меню *Tools*

Название на английском	Перевод на русский	Назначение
Tasks & Contexts	Задачи и контексты	
Save File as Template	Сохранить файл как шаблон	
IDE Scripting Console	Консоль сценариев IDE	Используется для написания простых скриптов, автоматизирующих работу функций PyCharm и извлекающих различную информацию

Окончание табл. 3.14

Название на английском	Перевод на русский	Назначение
Analyze Stack Trace	Анализ отслеживания стека	
Python Console	Консоль Python	Консоль Python позволяет выполнять команды и скрипты Python построчно
Create setup.py	Создать файл setup.py	
Convert to Jupyter Notebook	Преобразование в файл Jupyter Notebook	

Меню VCS

Меню VCS (Управление версиями) представлено на рис. 3.18.

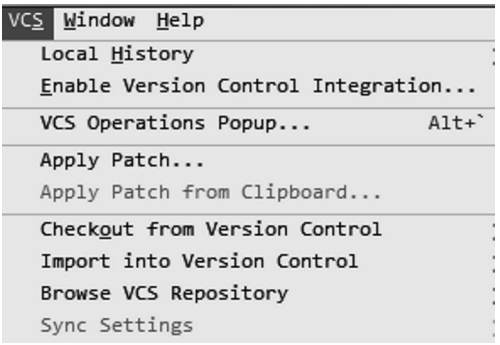


Рис. 3.18. Меню *VCS*

В табл. 3.15 показаны перевод и назначение основных функций меню VCS.

Таблица 3.15

Функции меню *VCS*

Название на английском	Перевод на русский	Назначение
Enable Version Control Integration	Включить интеграцию контроля версий	
VCS Operations Popup	Всплывающее окно операций	Для быстрого вызова любых команд, связанных с VCS
VCS Apply Patch	Применить патч	
Apply Patch from Clipboard	Применить патч из Clipboard	

Окончание табл. 3.15

Название на английском	Перевод на русский	Назначение
Checkout from Version Control	Проверка из системы контроля версий	
Import into Version Control	Импорт в систему контроля версий	
Browse VCS Repository	Просмотр репозитория VCS	
Sync Settings	Настройки синхронизации	

Меню Window

Меню Window представлено на рис. 3.19.

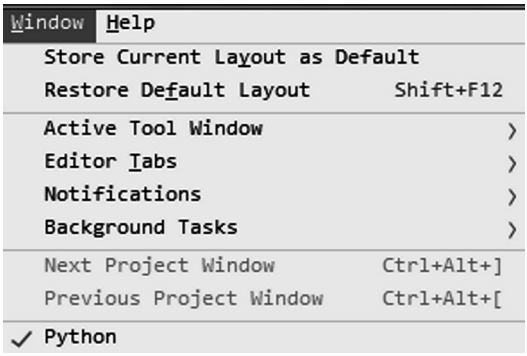


Рис. 3.19. Окно *Window*

В табл. 3.16 показаны перевод и назначение основных функций меню Window.

Таблица 3.16

Функции меню *Window*

Название на английском	Перевод на русский	Назначение
Store Current Layout as Default	Сохранить текущий макет по умолчанию	
Restore Default Layout	Восстановление макета по умолчанию	
Active Tool Window	Окно активного инструмента	Для отображения инструментов
Editor Tabs	Вкладки редактора	

Окончание табл. 3.16

Название на английском	Перевод на русский	Назначение
Notifications	Уведомления	Для включения и отключения уведомлений об определенных событиях, изменения их представления и, по желанию, включения их протоколирования
Background Tasks	Фоновые задачи	Выполнение задач в фоновом режиме
Next Project Window	Окно следующего проекта	
Previous Project Window	Окно предыдущего проекта	

Меню Help

Меню Help представлено на рис. 3.20.

В табл. 3.17 показаны перевод и назначение основных функций меню Help.

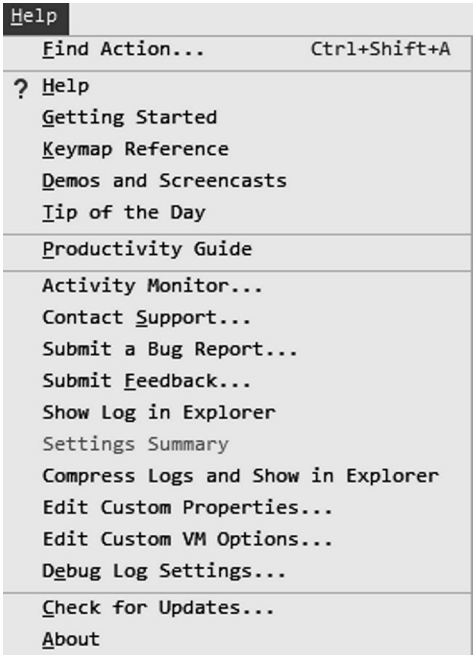


Рис. 3.20. Меню *Help*

Таблица 3.17

Функции меню *Help*

Название на английском	Перевод на русский	Назначение
Find Action	Найти действие	Если ваша клавиатура не имеет английской раскладки, PyCharm может неправильно определить все сочетания клавиш
Help	Помощь	
Getting Started	Начало работы	
Keymap Reference	Ссылка на комбинации клавиш	Вы можете увидеть сочетания клавиш для той карты клавиш, которую вы используете в IDE
Demos and Screencasts	Демоверсии и экранные записи	
Tip of the Day	Совет дня	
Productivity Guide	Справочник по продуктивности	
Activity Monitor	Монитор активности	В случае проблем с производительностью вы можете использовать Activity Monitor для отслеживания процента CPU
Contract Support	Поддержка контрактов	
Submit a Bug Report	Отправить сообщение об ошибке	
Submit Feedback	Отправить отзыв	
Show Log in Explorer	Показать лаг в проводнике	
Settings Summary	Краткое описание настроек	
Compress Logs and Show in Explorer	Сжать лаги и показать в проводнике	
Edit Custom Properties	Редактирование пользовательских свойств	
Edit Custom VM Options	Редактирование пользовательских параметров виртуальной машины	
Debug Log Settings	Настройки лагов отладки	

Окончание табл. 3.17

Название на английском	Перевод на русский	Назначение
Check for Updates About	Проверка обновлений О сайте	

4 Горячие клавиши PyCharm

PyCharm включает в себя различные сочетания клавиш для отображения наиболее часто используемых команд в редакторе. В этом разделе подробно рассматриваются раскладки клавиш. Вы можете найти список раскладок клавиш, доступных в меню **Help → Keymap Reference**, как показано на рис. 4.1.

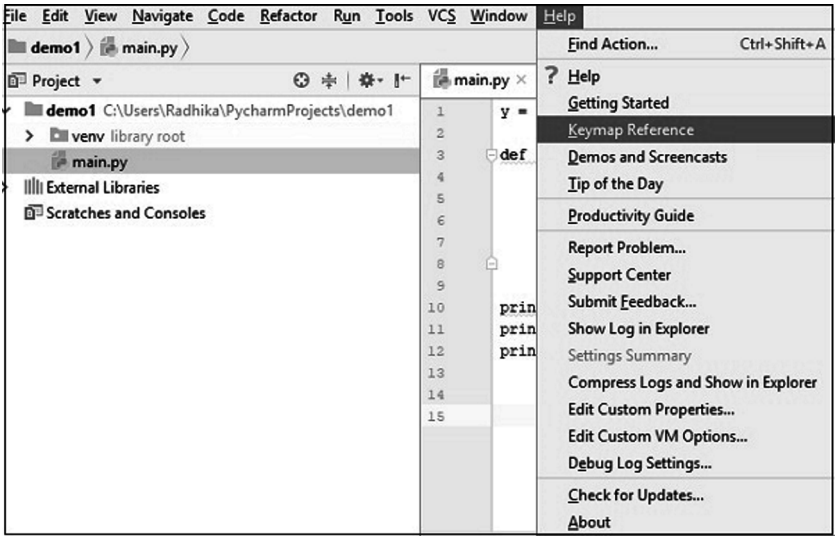


Рис. 4.1. Меню *Help*

На рис. 4.2 представлен перечень горячих клавиш в среде PyCharm.

В табл. 4.1–4.5 представлены переводы назначений горячих клавиш соответствующей темы.

Find any action inside the IDE	**Ctrl+Shift+A**
CREATE AND EDIT	
Show intention actions	**Alt+Enter**
Basic code completion	**Ctrl+Space**
Smart code completion	**Ctrl+Shift+Space**
Type name completion	**Ctrl+Alt+Space**
Complete statement	**Ctrl+Shift+Enter**
Parameter information/context info	**Ctrl+P**/**Alt+Q**
Quick definition	**Ctrl+Shift+I**
Quick/external documentation	**Ctrl+Q**/**Shift+F1**
Generate code	**Alt+Insert**
Override/implement members	**Ctrl+O**/**Ctrl+I**
Surround with...	**Ctrl+Alt+T**
Comment with line comment	**Ctrl+/**
Extend/shrink selection	**Ctrl+W**/**Ctrl+Shift+W**
Optimize imports	**Ctrl+Alt+O**
Auto-indent lines	**Ctrl+Alt+I**
Cut/Copy/Paste	**Ctrl+X**/**Ctrl+C**/**Ctrl+V**
Copy document path	**Ctrl+Shift+C**
Paste from clipboard history	**Ctrl+Shift+V**
Duplicate current line or selection	**Ctrl+D**
Move line up/down	**Ctrl+Shift+Up**/**Down**
Delete line at caret	**Ctrl+Y**
Join/split line	**Ctrl+Shift+J**/**Ctrl+Enter**
Start new line	**Shift+Enter**
Toggle case	**Ctrl+Shift+U**
Expand/collapse code block	**Ctrl+NumPad+**/**-**
Expand/collapse all	**Ctrl+NumPad+**/**-**
Save all	**Ctrl+S**
VERSION CONTROL	
VCS operations popup...	**Alt+`**
Commit	**Ctrl+K**
Update project	**Ctrl+T**
Recent changes	**Alt+Shift+C**
Revert	**Ctrl+Alt+Z**
Push...	**Ctrl+Shift+K**
Next/previous change	**Ctrl+Alt+Shift+Down**/**Up**
MASTER YOUR IDE	
Find action...	**Ctrl+Shift+A**
Open a tool window	**Alt+[0-9]**
Synchronize	**Ctrl+Alt+Y**
Quick switch scheme...	**Ctrl+`**
Settings...	**Ctrl+Alt+S**
Jump to source/navigation bar	**F4**/**Alt+Home**
Jump to last tool window	**F12**
Hide active/all tool windows	**Shift+Esc**/**Ctrl+Shift+F12**
Go to next/previous editor tab	**Alt+Right**/**Alt+Left**
Go to editor (from a tool window)	**Esc**
Close active tab/window	**Ctrl+Shift+F4**/**Ctrl+F4**

Рис. 4.2. Горячие клавиши среды PyCharm

FIND EVERYTHING	
Search everywhere	Double Shift
Find/replace	Ctrl+F/R
Find in path/Replace in path	Ctrl+Shift+F/R
Next/previous occurence	F3/Shift+F3
Find word at caret	Ctrl+F3
Go to class/file	Ctrl+N/Ctrl+Shift+N
Go to file member	Ctrl+F12
Go to symbol	Ctrl+Alt+Shift+N

NAVIGATE FROM SYMBOLS	
Declaration	Ctrl+B
Type declaration (JavaScript only)	Ctrl+Shift+B
Super method	Ctrl+U
Implementation(s)	Ctrl+Alt+B
Find usages/Find usages in file	Alt+F7/Ctrl+F7
Highlight usages in file	Ctrl+Shift+F7
Show usages	Ctrl+Alt+F7

REFACTOR AND CLEAN UP	
Refactor this...	Ctrl+Alt+Shift+T
Copy.../Move...	F5/F6
Safe delete...	Alt+Delete
Rename...	Shift+F6
Change signature...	Ctrl+F6
Inline...	Ctrl+Alt+N
Extract method	Ctrl+Alt+M
Introduce variable/parameter	Ctrl+Alt+V/P
Introduce field/constant	Ctrl+Alt+F/C
Reformat code	Ctrl+Alt+L

ANALYZE AND EXPLORE	
Show error description	Ctrl+F1
Next/previous highlighted error	F2/Shift+F2
Run inspection by name...	Ctrl+Alt+Shift+I
Type/call hierarchy	Ctrl+H/Ctrl+Alt+H

NAVIGATE IN CONTEXT	
Select in...	Alt+F1
Recently viewed/changed files	Ctrl+E/Ctrl+Shift+E
Last edit location	Ctrl+Shift+Back
Navigate back/forward	Ctrl+Alt+Left/Right
Go to previous/next method	Alt+Up/Down
Line/Column...	Ctrl+G
Go to code block end/start	Ctrl+]/[
Add to favorites	Alt+Shift+F
Toggle bookmark	F11
Toggle bookmark with mnemonic	Ctrl+F11
Go to numbered bookmark	Ctrl+[0-9]
Show bookmarks	Shift+F11

Рис. 4.2. Горячие клавиши среды PyCharm (продолжение)

BUILD, RUN, AND DEBUG	
Run context configuration	Ctrl+Shift+F10
Run/debug selected configuration	Alt+Shift+F10/F9
Run/debug current configuration	Shift+F10/F9
Step over/into	F8/F7
Smart step into	Shift+F7
Step out	Shift+F8
Run to cursor/Force run to cursor	Alt+F9/Ctrl+Alt+F9
Show execution point	Alt+F10
Evaluate expression...	Alt+F8
Stop	Ctrl+F2
Stop background processes...	Ctrl+Shift+F2
Resume program	F9
Toggle line breakpoint	Ctrl+F8
Toggle temporary line breakpoint	Ctrl+Alt+Shift+F8
Edit/view breakpoint	Ctrl+Shift+F8

Рис. 4.2. Горячие клавиши среды PyCharm (окончание)

Таблица 4.1

Работа с закладками

Поставить или снять закладку	F11
Аналогично с присвоением буквы или цифры	Ctrl+F11
Переход к закладке (удаление — клавишей Delete)	Shift+F11
Быстрый переход к закладке с присвоенным числом	Ctrl+Число

Таблица 4.2

Редактирование

Отменить последнее действие	Ctrl+Z
Отменить последнюю отмену действия	Ctrl+Shift+Z
Расширенная вставка из буфера обмена (с историей)	Ctrl+Shift+V
Инкрементальное выделение выражения	Ctrl (+Shift)+W
Перемещение между словами	Ctrl+влево/вправо
Прокрутка кода без изменения позиции курсора	Ctrl+вверх/вниз
Переход в начало/конец файла	Ctrl+Home/End
Удаление строки, отличие в том, где потом окажется курсор	Shift+Del (Ctrl+Y)
Удалить от текущей позиции до конца слова	Ctrl+Del
Удалить от текущей позиции до начала слова	Ctrl+Backspace
Дублировать текущую строку	Ctrl+D
Увеличить/уменьшить текущий отступ	Tab/Shift+Tab
Выравнивание отступов в коде	Ctrl+Alt+I
Приведение кода в соответствие Code-Style	Ctrl+Alt+L
Закомментировать/раскомментировать текущую строку	Ctrl+/

Окончание табл. 4.2

Закомментировать/раскомментировать выделенный код	Ctrl+Shift+/
Фолдинг, свернуть/развернуть	Ctrl+-/+
Фолдинг, свернуть/развернуть все	Ctrl+Shift+-/+
Сделать текущий скоуп сворачиваемым и свернуть его	Ctrl+Shift+
Сделать текущий скоуп несворачиваемым	Ctrl+
Замена в тексте	Ctrl+R
Замена во всех файлах	Ctrl+Shift+R

Таблица 4.3

Работа с окнами, вкладками

Перемещение между вкладками	Alt+влево/вправо
Закрыть вкладку	Ctrl+F4
Открытие/закрытие окон Project, Structure, Changes и т. д.	Altl+число
Переключение между вкладками и окнами	Ctrl+Tab
Закрыть активное окно	Shift+Esc
Открыть последнее закрытое окно	F12
Zoom, если он был вами настроен	Ctrl+колесико мыши

Таблица 4.4

Работа с поиском

Быстрый поиск по всему проекту	Дважды Shift
Быстрый поиск по настройкам, действиям и т. д.	Ctr+Shift+A
Перейти к следующему/предыдущему методу	Alt+вниз/вверх
Перемещение к началу и концу текущего скоупа	Ctrl+[и Ctrl+]
Поиск в файле	Ctrl+F
Поиск по всем файлам (переход — F4)	Ctr+Shift+F
Искать слово под курсором	Ctrl+F3
Искать вперед/назад	F3/Shift+F3
Переход к строке или строке:номеру_символа	Ctrl+G
Список методов с переходом к их объявлению	Ctrl+F12
Список недавно открытых файлов с переходом к ним	Ctrl+E
Список недавно измененных файлов с переходом к ним	Ctrl+Shift+E
Иерархия наследования текущего класса и переход по ней	Ctrl+H
Иерархия вызовов выбранного метода	Ctrl+Alt+H
Поиск класса по имени и переход к нему	Ctrl+N
Поиск файла по имени и переход к нему	Ctrl+Shift+N
Перейти к объявлению переменной, класса, метода	Ctrl+B

Окончание табл. 4.4

Перейти к реализации	Ctrl+Alt+B
Определить тип и перейти к его реализации	Ctrl+Shift+B
Перемещение назад по стеку поиска	Shift+Alt+влево
Перемещение вперед по стеку поиска	Shift+Alt+вправо
Переход к следующей/предыдущей ошибке	F2/Shift+F2
Найти все места, где используется метод/переменная	Shift+Alt+7
Как предыдущий пункт, только во всплывающем окне	Ctrl+Alt+7

Таблица 4.5

Генерация кода и рефакторинг

Полный автокомплит	Ctrl+Space
Автокомплит с фильтрацией по подходящему типу	Ctrl+Shift+Space
Простой автокомплит по словам, встречающимся в проекте	Alt+/
Реализовать интерфейс	Ctrl+I
Переопределить метод родительского класса	Ctrl+O
Генерация шаблонного кода (обход по итератору и т. д.)	Ctrl+J
Обернуть выделенный код в один из шаблонов	Ctrl+Alt+J
Генератор кода — сеттеров, зависимостей в pom.xml и т. д.	Alt+Insert
Переименование переменной, класса и т. д. во всем коде	Shift+F6
Изменение сигнатуры метода во всем коде	Ctrl+F6
Перемещение метода, класса или пакета	F6
Создать копию класса, файла или каталога	F5
Создать копию класса в том же пакете	Shift+F5
Безопасное удаление класса, метода или атрибута	Alt+Delete
Выделение метода	Ctrl+Alt+M
Выделение переменной	Ctrl+Alt+V
Выделение атрибута	Ctrl+Alt+F
Выделение константы (public final static)	Ctrl+Alt+C
Выделение аргумента метода	Ctrl+Alt+P
Инлайнинг метода, переменной, аргумента или константы	Ctrl+Alt+N
Оптимизация импортов	Ctrl+Alt+O

5 Основные элементы и структура программы на языке Python

Программа (программный код) на языке Python состоит из модулей, которые, в свою очередь, состоят из инструкций. Сами инструкции могут содержать выражения. Инструкции представляют собой указания компьютеру, определяющие операции, которые выполняет компьютер над данными. Общая структура программного кода на Python приведена рис. 5.1.

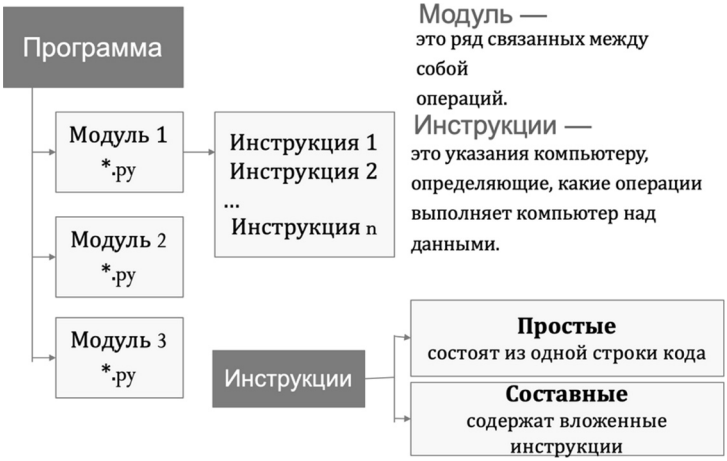


Рис. 5.1. Структура программы на языке Python

При написании программы программист может использовать только символы, которые допускается применять для записи программы. Множество допустимых символов называется *алфавитом языка программирования*. Алфавит языка Python включает следующие группы символов:

- латинские прописные (заглавные) буквы
  A, B, C, . . ., X, Y, Z;
- латинские строчные (малые) буквы
  a, b, c, . . . , x, y, z;
- арабские цифры
  0, 1, 2, . . . , 7, 8, 9;

- специальные символы (знаки арифметические, препинания, скобки и другие).

Минимальная единица программного кода, которая имеет определенный смысл для интерпретатора и не может быть разбита на более мелкие логические части, называется *лексемой*. Лексемы включают в себя следующие конструкции:

- комментарии;
- литералы;
- знаки пунктуации;
- идентификаторы;
- ключевые слова.

Комментарии

Комментарии предназначены для пояснения выполняемого кода с точки зрения разработчика. Транслятор Python пропускает комментарии, начиная с символы # до конца строки. Пример комментариев:

```
# Вся эта строка является комментарием
x = 5 # Это inline-комментарий (отступ 2 пробела)
y = '#это строка символов' # А это комментарий
```

Комментарии должны объяснять, почему что-то реализовано таким образом, а не иначе. Кроме того, указываются следующие моменты:

- предположения, важные решения и важные детали;
- проблемы, которые решает данная программа;
- проблемы, которые необходимо избегать и т. д.

Литералы

Литералами являются значения, представленные в коде программы (это числа или строки). Литерал является простейшим способом создания объекта. Литералы считаются «сырыми» данными, которые хранятся в строке или константе. Python поддерживает несколько типов литералов:

- строковые;
- числовые;
- логические;
- специальные.

Строковые литералы могут быть двух типов: *однострочные* и *многострочные*. Строковые литералы можно сформировать, заключив текст в кавычки (в одинарные или двойные, например 'мама', "123456"). Однострочные строки закачиваются

одной строкой программного кода. Многострочная строка представляет собой фрагмент текста, состоящий из нескольких строк программы. Существует два способа создания многострочной строки. В первом случае в конце каждой строки добавляется черная (обратная) косая черта \. Например,

```
mytext1 = 'Язык программирования
\ Питон'
```

В результате создается строка

```
'Язык программирования Питон'
```

Второй способ предусматривает использование тройных кавычек, например:

```
mytext2 = '''Язык
программирования
Питон'''
print(text2)
```

На консоль выводится строка

```
Язык
программирования
Питон
```

Числовые литералы неизменяемы в программном коде (в процессе выполнения программы) и могут принадлежать следующим различным числовым типам.

Целые числа (int) со знаком (целые числа могут быть двоичными (0b100), десятичными (-103), восьмеричными (0o12345) и шестнадцатеричным (0x13abef4)). Первый символ в написании числа — ноль.

Вещественные числа (float) имеют целую и дробную часть. Вещественные числа могут быть представлены в обычной (стандартной) форме (например, 2345.78) и в полулогарифмической форме, которая содержит букву «e», имеющее значение «десять в степени» (например, 23.71e-23).

Комплексные числа (complex) представляются в виде a+bj, где a — действительная часть комплексного числа; b — мнимая часть комплексного числа (например, 1.23+345j).

Ниже приведены примеры объявления числовых литералов.

```
x = 0b101100        # Binary Literal (двоичный литерал)
y = 100             # Decimal Literal (десятичный литерал)
z = 0o216           # Octal Literal (восьмеричный литерал)
u = 0x25a3f         # Hexadecimal Literal (шестнадцате-
```

```
                    # ричный литерал)
float_1 = 100.101   # вещественное число
float_2 = -1.5e2    # вещественное число
                    # в полулогарифмической форме
a = 5+3.14j         # Complex Literal (комплексный,
                    # сложный литерал
```

Логические литералы могут иметь одно из двух значений False и True. Примеры логических литералов:

```
x = (1 == True)
y = (3 == False)
a = 10+True
b = False+10
print('x is ', x)
print('y is ', y)
print('a:  ', a)
print('b:  ', b)
```

Выход:

```
x is TRUE
y is False
a:  11
b:  10
```

В языке программирования Python используется только один *специальный литерал* None. Литерал None используется для указания того поля, которое не создается.

Знаки пунктуации. Базовые операторы

Алфавит языка Python использует достаточное количество знаков пунктуации, которые используются для различных целей. Например, знаки «+» (плюс) или «–» (минус) могут использоваться для обозначения операторов сложения и вычитания, а знак запятой «,» используется для разделения списка параметров. Пусть задано выражение «4 + 5 равно 9». В данном выражении 4 и 5 суть операнды (operands) выражения, а символ «+» (плюс) называется оператором (operator). Большинство строк программного кода представляют собой выражения. После того как интерпретатор вычислит выражение, полученное значение можно присвоить некоторой переменной. Язык Python поддерживает различные виды операторов, которые используются в выражениях:

- арифметические операторы;

- операторы сравнения;
- операторы присваивания;
- логические операторы;
- битовые операторы;
- операторы принадлежности;
- операторы тождественности.

Оператор указывает на то, какое действие необходимо выполнить с операндами (или операндом, если оператор является унарным). В языке Python можно использовать много операторов, которые являются ключевыми словами или специальными символами.

Арифметические операторы

Перечислим арифметические операторы, которые можно использовать в программном коде, с указанием выполняемого действия с помощью данного оператора:

+ оператор суммы;
- оператор разности:
* оператор произведения (умножения);
/ оператор деления;
% оператор получения остатка от целочисленного деления;
** оператор возведения в степень;
// оператор целочисленного деления;
abs(a) модуль числа a;
divmod(a,b) получение пары чисел (a // b, a % b), например

```
divmod(25, 3) = (8,1);
```

pow(a, b[, c]) вычисляется a в степени b; если указано число c, то вычисляется остаток отделения на число c, например:

```
pow(2, 3) = 8, pow(4, 3, 5) = 4.
```

Операторы сравнения

== Проверяет, равны ли операнды между собой. Если они равны, то выражение становится истинным, т. е. принимает значение True.

|= Операция сравнения, обратная оператору ==. Проверяет, равны ли операнды между собой. Если не равны, то выражение становится истинным (значение True), в противном случае равно False.

> Проверяет, больше ли левый операнд, чем правый, если больше, то выражение становится истинным.

`<` Проверяет, меньше ли левый операнд, чем правый, если меньше, то выражение становится истинным.

`>=` Проверяет, больше левый оператор, чем правый, или равен ему. Если больше или равен, то выражение становится истинным.

`<=` Проверяет, меньше левый оператор, чем правый, или равен ему. Если меньше или равен, то выражение становится True.

Замечание. Не рекомендуется сравнивать между собой два вещественных значения (при выполнении операторов ==, |=, >=, <=). В этом случае правильным будет сравнение двух вещественных значений в виде abs(a - b) < eps вместо a == b, где eps — абсолютная погрешность сравнения двух вещественных значений (eps — очень маленькое значение, сопоставимое с ошибкой представления вещественного значения в памяти компьютера).

Операторы присваивания

`a = b` Присваивает значение правого операнда b (или значение выражения) левому операнду a.

`a += b` Эквивалентно записи a = a + b.

`a -= b` Эквивалентно записи a = a − b.

`a *= b` Эквивалентно записи a = a * b.

`a /= b` Эквивалентно записи a = a/b.

`a %= b` Делит значение левого операнда по остатку на правый операнд и присваивает левому. Эквивалентно записи a = a % b.

`a **= b` Эквивалентно записи a = a ** b.

`a //= b` Целочисленно делит значение левого операнда на правый операнд и присваивает левому. Эквивалентно записи a = a // b.

Замечание. Можно поменять значения у двух переменных без использования дополнительной (промежуточной) переменной с помощью команды a,b = b,a (значения переменных меняются местами).

Побитовые операторы

Побитовые операторы (bitwise) работают с битами данных и выполняют операции бит за битом. Данные операторы применяются для целых чисел, которые предварительно преобразуются в двоичный формат, затем операции выполняются побитно, а

окончательный результат возвращается в десятичном формате. В Python существует шесть побитовых операторов.

- & Побитовый оператор AND (логическое И).
- | Побитовый оператор OR (логическое ИЛИ).
- ∧ Побитовый оператор XOR (исключающее ИЛИ, сложение по модулю 2).
- ∼ Комплиментарный оператор.
- << Побитовый сдвиг влево (Left Shift).
- >> Побитовый сдвиг вправо (Right Shift).

Рассмотрим пример, в котором a = 60 и b = 13. В бинарном формате они имеют следующее представление:

a = 0011 1100
b = 0000 1101

Применим побитовые операции:

```
a & b = 0000 1100
a | b = 0011 1101
a ∧ b = 0011 0001
```

Побитовое выполнение операций определяется следующими правилами над отдельными битами:

Побитовая операция AND	Побитовая операция OR	Побитовая операция XOR
0 & 0 = 0	0 | 0 = 0	0 ∧ 0 = 0
0 & 1 = 0	0 | 1 = 1	0 ∧ 1 = 1
1 & 0 = 0	1 | 0 =1	1 ∧ 0 = 1
1 & 1 = 1	1 | 1 = 1	1 ∧ 1 = 0

Комплиментарный оператор (Bitwise Ones'Complement Operator) выполняется по правилу дополнения числа a в соответствии с формулой −(a+1), например:

```
a = 60 = 00111100 (Binary)
∼a = ∼00111100
    = -(00111100+1)
    = -(00111101)
    = -61 (Decimal)
```

Операторы Left Shift и Right Shift сдвигают двоичную последовательность влево и вправо на указанное число позиций соответственно, например:

```
1010 << 3 = 1010000 = 80 (Decimal)
1010 >> 2 = 0010 = 10 = 2 (Decimal)
```

Логические операторы

Логические операторы применимы только к операндам (константам, переменным, функциям) логического типа и возвращают логическое значение False или True. В языке Python определены следующие операторы: not — логическое отрицание (инверсия, оператор НЕ); and — логическое умножение (конъюнкция, оператор И); or — логическое сложение (дизъюнкция, оператор ИЛИ). Выполнение логических операторов осуществляется в соответствии с правилами:

Оператор НЕ	Оператор И	Оператор ИЛИ
not False = True	False and False = False	False or False = False
not True = False	False and True = False	False or True = True
	True and False = False	True or False = True
	True and True = True	True or True = True

Операторы принадлежности

В программе на языке Python можно применять операторы членства (принадлежности) некоторого элемента последовательности (строки, списка или кортежа). Таких оператора два. Оператор in выдает значение истина (True), если находит переменную в заданной последовательности, и значение ложь (False) в противном случае. Оператор not in выдает значение истина (True), если не находит переменную в заданной последовательности, и выдает значение ложь (False) в противном случае.

Операторы тождественности

Операторы тождественности или идентичности сравнивают расположение двух объектов в памяти компьютера. Таких операторов два:

- is — значение истина (True), если переменные по стороны от оператора указывают на один объект, и значение ложь (False) в противном случае;
- is not — значение ложь (False), если переменные по обе стороны от оператора указывают на один объект, и значение истина (True) в противном случае.

Приоритет выполнения операторов

Иногда выражение содержит несколько операторов. В этом случае приоритет оператора используется для определения порядка их выполнения. Можно создать группу выражений, используя круглые скобки, выражение в скобках сначала вычисля-

ется, прежде чем оно сможет участвовать в дальнейших вычислениях. Некоторые операторы имеют одинаковый уровень приоритета, в этом случае выражение выполняется слева направо. Ниже перечислены приоритеты операторов в порядке убывания.

**	Возведение в степень
~	Комплементарные операции
*, /, //, %	Умножение, деление, деление по модулю, целочисленное деление
+, -	Сложение, вычитание
<<, >>	Операторы сдвига вправо и влево
&	Бинарное И
|, ^	Бинарное ИЛИ и бинарное исключающее ИЛИ
==, |=, >, <, >=, <=	Операции сравнения (равно, не равно, больше, меньше, больше или равно, меньше или равно)
=, +=, -=, *=, /=, //=, %=	Присваивание
is, is not	Тождество и не тождество
in, not in	Принадлежит и не принадлежит множеству
not, and, or	Логическое отрицание, умножение и сложение

Ленивые логические выражения

Ленивые логические выражения являются особенностью языка Python. Разберем эти особенности на примерах.

Пусть переменная x имеет значение 12, а переменная y — значение False. При выводе выполняется выражение x or y:

```
x = 12
y = False
print(x or y) # выводится значение 12
```

В результате работы этого выражения на экране появится не значение True (как можно было бы ожидать), а число 12, так как Python начинает интерпретировать логическое выражение, видит, что x является истинным, и понимает, что ему не нужно выполнять оставшуюся часть логического выражения. Поэтому он не будет проводить дальнейшую проверку. Значение x — истина, а далее стоит операция or, тогда можно остановиться и результатом выполнения выражения будет значение x.

В другом примере значение переменной x присвоено значение 12, переменной z — строковое значение 'boom', поэтому в результате проверки операции and выводится значение boom.

```
x = 12z = 'boom'
print(x and z) # выведется значение boom
```

При реализации ленивого логического выражения проверка его происходит до тех пор, пока она имеет смысл, и результатом является последнее значение.

Идентификаторы

Идентификатор является обычным именем, которое дается ссылке на какой-либо объект. Любой идентификатор имеет некоторые правила и соглашения наименования:

- правила:
  - может состоять из букв (ASCII и Unicode), знаков подчеркивания или арабских цифр;
  - цифра не может быть на первом месте;
  - чувствителен к регистру (например, UserName, username и USERNAME разные идентификаторы);
  - не должен совпадать с каким-либо из ключевых слов языка Python;
- соглашения:
  - использовать змеиный_регистр (snake_case) — именование переменных и функций, когда название начинается с маленькой буквы, а слова разделяются знаком подчеркивания;
  - не использовать предопределенные имена;
  - не использовать два знака подчеркивания __ в начале и конце идентификатора, а также знак подчеркивания и двойное подчеркивание в чале идентификатора (данный синтаксис имеет специальное назначение).

Ключевые слова можно вывести с помощью простейшего программного кода:

```
import keywordprint(keyword.kwlist)
```

В результате получим список ключевых слов:

```
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break',
'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for',
'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not',
'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

Понятие предопределенных имен рассмотрим на следующем примере. Имеется встроенная функция dir(), которая возвращает список атрибутов объекта. Если эта функция вызывается без атрибутов, то она возвращает список встроенных атрибутов:

```
['__annotations__', '__builtins__', '__cached__', '__doc__', '__file__',
'__loader__', '__name__', '__package__', '__spec__']
```

С помощью оператора

```
print(dir(__builtins__))
```

можно вывести все предопределенные имена:

```
['ArithmeticError', 'AssertionError',..., 'vars', 'zip']
```

Синтаксическая структура программы

Для составления программного кода необходимо объединить лексемы в синтаксические конструкции. Примеры синтаксических конструкций приведены в следующем примере.

```
# 1. Простейшие синтаксические конструкции — литералы:
# простые
5                   # Литерал — целое число
'string'            # Литерал — строка
# составные
[1, 2, 3]           # Литерал — список чисел
                    # с использованием знаков операций
x = 23              # Присваивание
x += 1              # Увеличение на 1
# 2. Операторы
if x > 10:          # Если выражение истинно,
x += 1              # то увеличить x на 1
print(x)            # Вывести значение x (x = 25)
```

Синтаксические конструкции могут вкладываться друг в друга (выполняться внутри других конструкция, т. е. принадлежать им), образуя блочную структуру. Блок кода начинается двоеточием, а тело блока выделяется обязательным отступом в виде 4 пробелов. Методология структурного программирования предусматривает представление программы в виде иерархической структуры блоков. Блоки, в свою очередь, могут состоять из таких же блоков. В языке Python можно выделить следующие виды блоки (которые являются составными элементами программы):

- **последовательность (инструкция)** — любое атомарное действие, например присваивание;

- **ветвление (условие)** — выполнение инструкции в зависимости от определенного условия;
- **цикл** — многократное исполнение набора инструкций;
- **подпрограмма (процедура или функция)** — часть компьютерной программы, содержащая описание определенного набора инструкций, которая может быть многократно вызвана из разных частей программы (может содержать инструкции, ветвления, циклы);
- **класс** — абстрактный тип данных в объектно-ориентированном программировании, задающий общее поведение для группы объектов — модель объекта (может содержать все перечисленные выше блоки);
- **модуль** — функционально законченный фрагмент программы, оформленный в виде отдельного файла с исходным кодом или поименованной непрерывной ее части и может содержать все перечисленные выше блоки);
- **пакет** — логически законченная совокупность модулей как единое целое.

Инструкции ввода и вывода данных

Любая программа составляется и реализуется с целью получения требуемого результата, поэтому в программном коде предусмотрена специальная функция выводов результатов и промежуточных данных print(). С другой стороны, чтобы получить требуемый результат по разработанной алгоритму (программе), необходимо задать исходные данные. Для этих целей можно использовать инструкцию присваивания, однако подобные действия нарушают важное свойство любого алгоритма — массовость. Поэтому для ввода исходных данных применяют специальную встроенную функцию input(). Совместное применение функций print() и input() обеспечивают организацию удобного интерфейса между пользователем и программой.

Использование функции input(). Встроенная функция input() позволяет получить ввод пользователя в виде строки

```
input('введите значение переменной x = ')
```

В процессе выполнения данной инструкции в поле терминала появится сообщение-приглашение для ввода значения переменной x в формате

```
'введите значение переменной x =
```

После вывода данного сообщения ожидается ввод пользователем конкретного значения. Ввод подтверждается нажатием

клавиши Enter. Чтобы введенное значение присвоилось переменной x, необходимо инструкцию ввода записать в форме

```
x = input('введите значение переменной x = ')
```

Необходимо помнить, что функция input() возвращает введенное значение типа строки символов (string), так как ввод/вывод данных в Python реализуется с помощью стандартных текстовых файлов. При необходимости ввода, например, числовых данных нужно использовать явное приведение типов:

```
x = float(input('введите значение x = '))
```

Пусть инструкция `x = input()` обеспечивает ввод значения 10; `x = '10'` — тип string; тогда инструкция `x = float(input())` позволяет получить после ввода 10 значение `x = 10.0` — тип float.

Использование функции print(). Встроенная функция print() предназначена для вывода полученных результатов на консоль (экран монитора). Формат оператора input() в общем случае имеет вид:

```
print(value1, ..., [valueN], [sep = ' '],
    [end = '\n'],[file = sys.stdout)],
    [flus h = False])
```

где указанные в скобках параметры имеют следующий смысл (квадратные скобки указывают на необязательность параметра):

- value1, ..., valueN — список выводимых значений любого типа (при печати все объекты преобразуются в строки);
- sep — разделитель между выводимыми значениями, по умолчанию одиночный пробел;
- end — строка, завершающая вывод, по умолчанию — символ переноса строки вывода;
- file — выходной поток (выходной файл), по умолчанию — консоль (экран монитора);
- flus — флаг, определяющий режим вывода в файл при наличии цикла; действует только в том случае, если выбран файл в аргументе file; при значении True вывод в файл происходит незамедлительно при вызове функции print(). При значении False (по умолчанию) вывод будет произведен только после выхода из цикла.

В простейшем случае обязательным параметром является только список объектов для вывода:

```
print(value1, ..., [valueN])
```

Поэтому на первоначальных стадиях изучения языка Python рекомендуется использовать упрощенную форму вывода результатов выполнения программы в следующей форме:

```
print('вывод результатов:')
print('x = ', x, 'name = ', name)
```

В том случае, когда оператор print используется без указания каких-либо параметров, ничего не выводится, а пропускается строка в выводимых результатах.

Модуль math

Для выполнения сложных вычислительных операций крайне полезными могут оказаться встроенные функции языка, реализованные в библиотеке math. Для того чтобы использовать функции модуля math, необходимо его подключить для использования в программе. Сделать это можно двумя способами.

Первый вариант. Использование команды **import** math. В этом случае необходимые функции в тексте программы, вызываются с помощью составного имени math.<имя_функции>, например:

```
import math
x = math.log(4.57)
print('x = ', x, 'z = ', math.ceil(84.56))
# вывод: x = 1.5195132049061133 z = 85
```

Второй вариант. Использование команды **from** math **import** <имя_функции>. При этом пропадает необходимость указывать имя модуля перед вызовом функции. Вместо имени функции может быть указан символ * для выбора любой функции модуля:

```
from math import *x = log(4.57)
print('x = ', x, 'z = ', ceil(84.56))
#вывод: x = 1.5195132049061133 z = 85
```

Ниже приведен полный список функций, входящих в модуль math. Имена функций представлены без приставки math.<имя_функции>.

ceil(X)	Округление значения X до целого вверх.
copysign(X, Y)	Возвращает число, имеющее модуль такой же, как и у числа X, а знак, как у числа Y.
fabs(X)	Модуль числа X, т. е. |X|.
factorial(X)	Факториал числа X т. е. X! (factorial(1) = 0).

floor(X)	Округление значения X до целого вниз.
fmod(X, Y)	Остаток от деления X на Y.
frexp(X)	Возвращает мантиссу и экспоненту числа X = = A*2^B. Применение: A, B = frexp(X).
ldexp(X, I)	Возвращает значение X * 2^I. Функция, обратная функции frexp()
fsum(a1,...,an)	Сумма всех членов последовательности a1, a2, ..., an. Эквивалент встроенной функции sum(), но fsum() более точна для чисел с плавающей точкой.
isfinite(X)	Является ли X числом.
isinf(X)	Возвращает True, если X является бесконечностью, в противном случае — False.
isnan(X)	Возвращает True, если X является NaN (Nol a Namber – не число), иначе — False.
modf(X)	Возвращает дробную и целую часть числа X.Оба числа имеют тот же знак, что X. Пример: modf(20.1) => (0.1, 20)
trunc(X)	Усекает значение X до целого.
exp(X)	e^X.
expm1(X)	$e^X - 1$. При малых X точнее, чем exp(X) – 1.
log(X, [base])	Логарифм X по основанию base. Если base не указан, то вычисляется натуральный логарифм.
log1p(X)	Натуральный логарифм (1+X). При малых X точнее, чем log(1+X)
log10(X)	Логарифм X по основанию 10.
log2(X)	Логарифм X по основанию 2.
pow(X, Y)	X^Y.
sqrt(X)	Квадратный корень из X. Альтернатива выражению X **(1/2), но X** (1/2) вернет комплексное число для отрицательного X, а sqrt(X) выдаст ошибку.
acos(X)	Арккосинус. X в радианах.
asin(X)	Арксинус. X в радианах.
atan(X)	Арктангенс. X в радианах.
atan2(Y, X)	Арктангенс Y/X. В радианах. С учетом четверти, в которой находится точка (X, Y).
cos(X)	Косинус X. X указывается в радианах.
sin(X)	Синус X. X указывается в радианах.

tan(X)	Тангенс X. X указывается в радианах.
hypot(X, Y)	Вычисляет гипотенузу треугольника с катетами X и Y. Равносильно sqrt(X * X + Y * Y).
degrees(X)	Конвертирует радианы в градусы.
radians(X)	Конвертирует градусы в радианы.
cosh(X)	Вычисляет гиперболический косинус.
sinh(X)	Вычисляет гиперболический синус.
tanh(X)	Вычисляет гиперболический тангенс.
acosh(X)	Вычисляет обратный гиперболический косинус.
asinh(X)	Вычисляет обратный гиперболический синус.
atanh(X)	Вычисляет обратный гиперболический тангенс.
erf(X)	Функция ошибок erf(X).
erfc(X)	Дополнительная функция ошибок (1 - erf(X)).
gamma(X)	Гамма-функция Γ(X).
lgamma(X)	Натуральный логарифм гамма-функции Γ(X).
pi	Число Архимеда 3,1415926...
e	Основание натуральных логарифмов 2,718281...

6 Отладка

Понятие об отладке программ

Отладка программы является итеративным процессом обнаружения и исправления ошибок и обычно требует последовательного выполнения четырех этапов:

1) выявление ошибки;
2) локализация ошибки в тексте программы;
3) установление причины ошибки;
4) исправление ошибки.

Некоторые ошибки проявляются после первого же запуска программы на выполнение, и для их обнаружения не надо прибегать ни к каким специальным средствам. Некоторые ошибки проявляются в случайные моменты работы программы. С такими ошибками справиться труднее всего — зафиксировать условия возникновения ошибки, понять причину ошибки и устранить ее.

В целях обнаружения подобных ошибок осуществляется **тестирование программы**, т. е. ее выполнение для специально подобранных представительных контрольных примеров — тестов.

Тест — это такой набор исходных данных, для которого вручную или другим способом просчитаны промежуточные и конечные результаты и который может быть использован для получения информации о надежности проверяемой программы.

Тестирование программы должно включать в себя прогон трех видов контрольных примеров: нормальных случаев, граничных случаев и случаев неправильных данных. *Нормальные случаи* — это примеры с правильными входными данными. Если программа не работает в подобных случаях, она требует серьезных переделок. Граничные контрольные примеры помогают установить, способна ли программа нормально реагировать на особые случаи во входных данных. *Граничные примеры* представляют собой данные, которые, будучи математически корректными, приводят программу к необходимости работать осо-

бым образом. *Неправильными* являются такие данные, которые расположены вне допустимого диапазона. Примеры с неправильными данными должны быть обработаны соответствующим образом, поскольку в повседневной эксплуатации программе придется иметь дело и с неверными входными данными.

После того как ошибка обнаружена, необходимо найти в исходном тексте программы то место, в котором она возникала, т. е. *локализовать ошибку.* Можно использовать ряд различных методов отладки, позволяющих обнаружить расположение ошибки; выбор существенно зависит от особенностей ситуации. Большинство программистов начинают с неформального метода, известного под названием *проверка за столом.* Используя контрольный пример, который привел к ошибке в программе, программист аналитически трассирует листинг программы в надежде локализовать ошибку. Проверка за столом — это хороший метод, поскольку он заставляет программиста детально понять работу программы. Если применение метода проверки за столом оказалось бесплодным, нужно использовать специальные методы и способы отладки, позволяющие наблюдать за передачей управления в программе и за изменением значений наиболее важных переменных. Полученная отладочная информация позволит локализовать подозрительные ситуации, провести анализ и выявить причину ошибки, устранить ее, а затем продолжить поиск других ошибок.

Причины и типы ошибок

В общем случае ошибки могут возникать на любом этапе разработки программы, причина ошибок может быть связана с недопониманием сути задачи, недостатками проектирования алгоритма, неправильным использованием языковых средств. При выполнении программы ошибки разного типа проявляют себя различным образом, и их принято подразделять на следующие группы:

- синтаксические ошибки;
- семантические ошибки;
- логические ошибки.

Синтаксические ошибки проявляются на этапе компиляции программы и возникают в связи с нарушением синтаксических правил написания предложений языка программирования. Если компилятор встречает в тексте программы оператор или

описание, которые он не может интерпретировать, то он позиционирует курсор на место обнаруженной ошибки и в строку статуса выводит сообщение, содержащее номер ошибки и ее краткое описание.

Семантические ошибки проявляются на этапе выполнения программы при ее попытке вычислить недопустимые значения параметров или выполнить недопустимые действия. Причина возникновения ошибок данного типа связана с нарушением семантических правил написания программ (примером являются ситуации попытки открыть несуществующий файл или выполнить деление на нуль). Если программа обнаруживает ошибку такого типа, то она завершает свое выполнение и выводит соответствующее сообщение в терминальном окне, содержащее номер строки с ошибкой, содержание строки и тип ошибки.

Логические (*смысловые*) *ошибки* — самые сложные и трудноуловимые, связанные с неправильным применением тех или иных алгоритмических конструкций. Эти ошибки при выполнении программы могут проявиться явно (выдано сообщение об ошибке, нет результата или выдан неверный результат, программа «зацикливается»), но чаще они проявляют себя только при определенных сочетаниях параметров или вообще не вызывают нарушения работы программы, которая в этом случае выдает правдоподобные, но неверные результаты.

Ошибки первого типа легко выявляются самим компилятором. Обычно устранение синтаксических ошибок не вызывает особых трудностей. Более сложно выявить ошибки второго и особенно третьего типа. Для обнаружения и устранения ошибок второго и третьего типа обычно применяют специальные способы и средства отладки программ.

Способы и средства отладки

В ходе отладки программа должна быть проверена в двух измерениях: в пространстве и во времени. Первое представляет собой контроль содержимого памяти в конкретные моменты работы программы, отслеживание текущих значений всех или выбранных групп переменных, проверку на соответствие их значений ожидаемым типам. Второе — это отслеживание хода выполнения алгоритма для проверки правильности заданной последовательности операций и передач управления при различных значениях параметров. Самым распространенным и полезным приемом отладки, позволяющим объединить обе формы контроля, являются отчеты о трассировке.

Трассировка программы — это регистрация логического пути выполнения программы — последовательности выполнения ее операторов/блоков с контрольной выдачей информации о результатах каждого шага — обо всех изменениях значений рабочих переменных и параметров связи. Сам принцип трассировки — слишком общий. На практике реализуют трассировку программы в том или ином объеме, используя различные способы и средства отладки.

Самый простой способ отладки — это расстановка в тексте программы *отладочных печатей* промежуточных результатов вычислений, позволяющих проследить *логический и арифметический следы* программы, т. е. каким образом она выполнялась и что она вычисляла. Отладочные печати ставятся в узловых/ключевых точках программы, позволяющих контролировать ошибки ввода (эхо-печать введенных данных), результаты вычислительных операций и логику работы программы или отдельных ее частей. Отладочные операторы оформляются в отдельные строки, выделяются особым образом (например, открывающим и закрывающим комментариями) и в зависимости от цели контроля могут содержать вывод значений контролируемых переменных, проверку условий или идентифицирующее сообщение о прохождении заданной точки программы, о начале или завершении работы определенного участка/блока. Средства для отладки могут быть вставлены в программу еще при ее разработке. В ходе отладки количество и место расположения отладочных операторов меняется, но их лучше не удалять, а превращать в комментарии. Такой способ отладки весьма трудоемок и может сам служить источником ошибок.

Процесс отладки значительно облегчается, если использовать для этого *системные средства отладки* — специальные программы — отладчики, имеющиеся в программном обеспечении компьютера.

Встроенный отладчик среды PyCharm (Debugger) позволяет контролировать ход выполнения программы — выполнять трассировку программы без изменения самой программы с помощью следующих действий:

- выполнение программы построчно/по шагам;
- остановка выполнения программы в заданной точке останова;
- перезапуск программы, не закончив ее выполнение;

- вычисление выражений на основе переменных в программе, а также просмотр значений этих переменных с помощью механизма наблюдателей;
- назначение и модификация значений любых переменных и параметров программы, а также получение некоторых дополнительных сведений о программе, например, списка активных процедур.

Эти возможности позволяют, отследив выполнение каждого оператора/операции, определить местоположение ошибки и понять ее причину. Далее рассмотрены перечисленные управляющие средства.

Отладка программ в среде PyCharm

В PyCharm имеется мощный встроенный отладчик, значительно упрощающий отладку программ. Основными инструментами отладки являются точки контрольного останова и окно наблюдения за переменными. Для запуска режима отладки необходимо выполнить команду *Run* → *Debug*.

Точки контрольного останова. Оператор в программе, перед выполнением которого программа прервет свою работу и управление будет передано среде PyCharm, определяет *точку контрольного останова.* Точка останова задается с помощью нажатия левой кнопки мыши слева от начала нужной строки. При этом перед строкой появится красная точка (рис. 6.1).

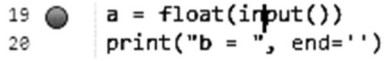


Рис. 6.1. Метка точки останова в редакторе кода

Окно точек останова (рис. 6.2) содержит список всех установленных в проекте точек, перед выполнением которых происходит прекращение работы программы и управление получает среда PyCharm. Вызов данного окна может быть произведен из меню *Run* → *View Breakpoints*. Также окно свойств для одной точки можно открыть нажатием правой кнопки мыши на точке в редакторе кода.

Для добавления новой точки следует нажать на кнопку «+» в верхней части окна.

Окно наблюдения. Наблюдать за состоянием переменной или выражения можно с помощью специального окна, которое появляется при начале отладки. По умолчанию содержимое окна

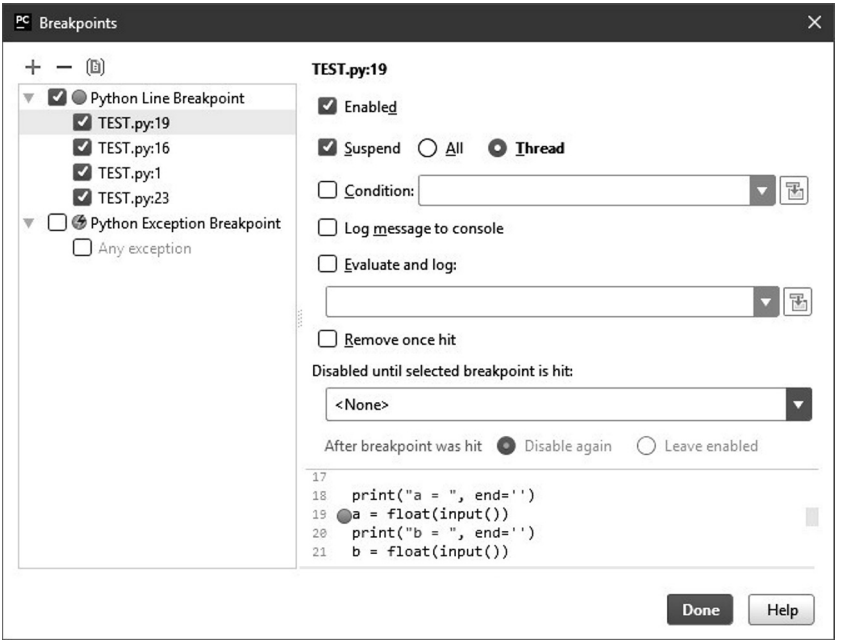


Рис. 6.2. Окно точек останова

наблюдателей (Watches) смешано с содержимым окна переменных (Variables). Для разделения этих окон необходимо нажать на кнопку с пиктограммой очков (рис. 6.3).

Окно наблюдения (рис. 6.4) используется в отладочном режиме для наблюдения за изменением значений выражений, помещенных в это окно.

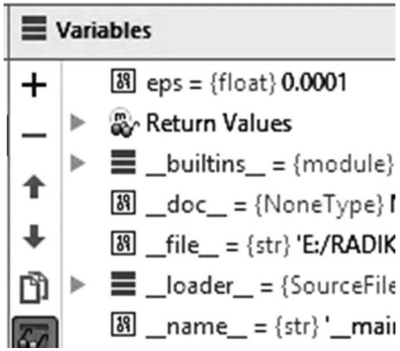


Рис. 6.3. Окно переменных программы

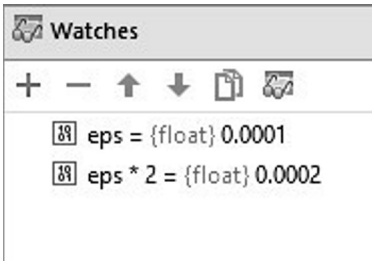


Рис. 6.4. Окно наблюдения

Для добавления нового выражения нужно выбрать опцию New Watch (кнопка «+» в верхней части окна). В появившейся строке ввести выражение.

Значения переменных также можно посмотреть во время останова программы, наведя курсор мыши на переменную в тексте кода. Кроме того, значения переменных в коде во время выполнения пишутся рядом с ними в виде псевдокомментариев.

Принудительное прерывание работы программы. Если программа запущена из среды PyCharm, ее работу можно прервать в любой момент опцией *Run* → *Pause Program* или установив точку контрольного останова в той части программы, которая выполняется в данный момент или будет выполнена.

Трассировка программы. Перед исполнением оператора, в котором установлена точка контрольного останова, работа программы будет прервана, управление получит среда PyCharm, а в окне наблюдения отразится текущее значение наблюдаемых переменных и (или) выражений. Теперь программист может прослеживать работу программы по шагам с помощью клавиш F10 и F11 или инструментальных кнопок. При нажатии клавиши F10 будут выполнены запрограммированные в текущей строке действия и работа программы прервется перед выполнением следующей строки текста программы.

Следует заметить, что контрольная точка останова выделяется по умолчанию красным цветом, а текущая прослеживаемая строка — синим.

Чтобы установить/снять точку контрольного останова, достаточно щелкнуть мышью по служебной зоне слева от нужной строки или установить в эту строку текстовый курсор и нажать клавишу F9.

При нажатии клавиши F11 выполняются те же действия, что и при нажатии клавиши F10, однако, если в текущей строке содержится вызов подпрограммы пользователя, программа прервет свою работу перед выполнением первого исполняемого оператора этой подпрограммы, т. е. клавиша F11 позволяет прослеживать работу вызываемых подпрограмм.

После трассировки нужного фрагмента программы можно продолжить нормальную ее работу, нажав клавишу F5.

Действия в точках прерывания. В PyCharm с любой точкой можно связать одно или несколько действий. Для этого нужно активизировать окно точек останова, затем выбрать нуж-

ную точку, после чего откроется список настроек для нее. Для точки определены свойства:

- Enabled (включение останова в этой точке);
- Suspend (All/Thread) (приостановка всего приложения или только текущего потока);
- Condition (условие останова);
- Log message to console (в терминальное окно будет выводиться сообщение о достижении данной точки);
- Evaluate and log (можно вычислить некоторое выражение и поместить его результат в сообщение из предыдущего пункта);
- Remove once hit (точка останова будет удалена по достижении. Осторожнее с этой опцией, при завершении отладки точка не появится и ее придется устанавливать заново).

Группировка точек прерывания. В PyCharm имеется возможность объединения точек останова в группы. Для этого используется все то же окно (см. рис. 6.2): для добавления точек в группу их можно переместить нажатием правой кнопки мыши и выбором команды *Move to group*. Там же находится команда создания новой группы *Create New*. Группу точек можно включать и выключать переключением общего флага слева от названия группы.

Вычисление выражений и изменение значений. С помощью окон Variables и Watches вкладки отладчика (рис. 6.5) можно узнать значение любого выражения или установить в переменную другое значение. Эти окна вызываются в режиме отладки автоматически.

В окне Variables можно изменить значение любой переменной, выделив ее и нажав клавишу F2. В окне Watches можно наблюдать за значениями нужных переменных, а также вычисляемых выражений. Для добавления переменной или выражения в наблюдатели достаточно нажать на кнопку «+» на экране, затем ввести выражение и подтвердить ввод клавишей Enter.

Ведение протокола работы программы. В ряде случаев бывает неудобно или невозможно пользоваться пошаговой отладкой программ. В таких ситуациях могут помочь контрольные точки, которые не прерывают работу программы, а лишь помещают некоторую информацию в специальный файл трассировки. Для реализации такой точки следует в ее свойствах отключить параметр Suspend и включить параметр Log message

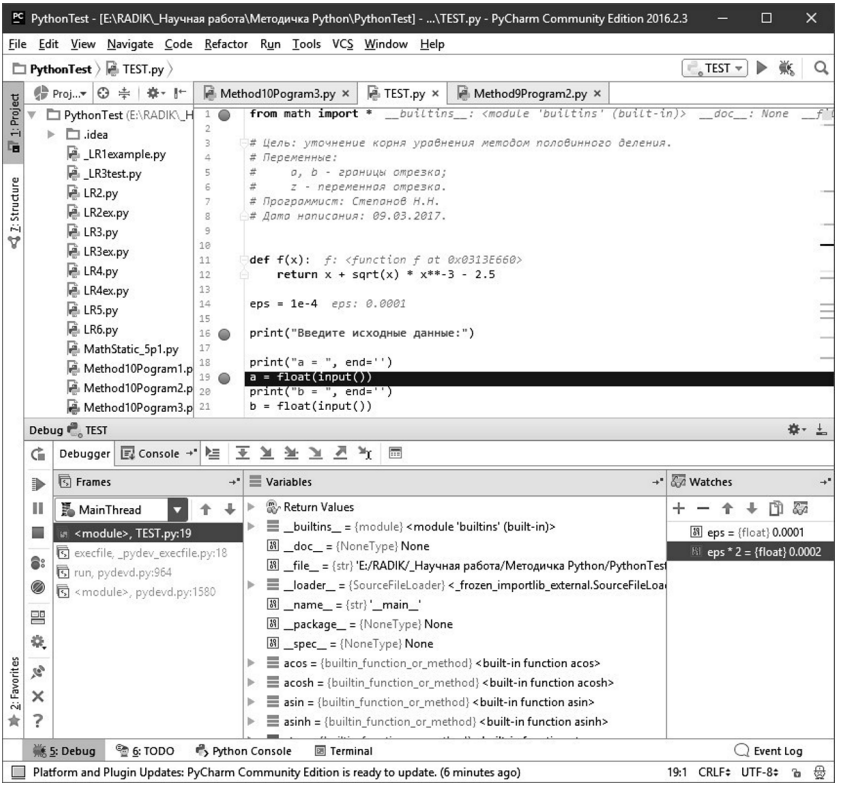


Рис. 6.5. Среда PyCharm в режиме отладки программы

to console (дополнительно — еще и Evaluate and log). Тогда отладочная информация будет выводиться в консоль без остановки выполнения программы.

7 Руководство PEP 8 по написанию кода

С целью создания качественного программного кода и унификации проектной документации программу на Python необходимо составлять в соответствии с требованиями руководства по написанию кода. Соответствующие правила определены в стандартах PEP (Python Enhancement Proposal). Эти стандарты обеспечивают создание унифицированной документации для утвержденных возможностей языка Python. Самый известный PEP имеет восьмой номер PEP 8 и содержит перечень принципов написания красивого, лаконичного и качественного кода.

PEP 8 создан на основе рекомендаций создателя Python Гвидо Ван Россума и постоянно обновляется и модернизируется. Основная идея Гвидо состоит в том, что программный код читается и просматривается намного больше раз, чем пишется. Поэтому рекомендации, направленные на улучшение читаемости кода, делают его согласованным между большим числом проектов. Указывают на причины, которые оправдывает нарушение этого правила:

- когда применение правила делает программный код менее читаемым;
- если происходит расширение уже существующего проекта, отличающегося по стилю (в силу исторических причин) от требований PEP 8.

Внешний вид кода

Внешний вид программного кода определяется такими факторами, как использование отступов, использование табуляции и пробелов, ограничение максимальной длины строки, использование пустых строк, кодировка исходного файла, применение импортов.

Рекомендуется использовать четыре пробела на каждый уровень *отступа.* При этом используйте *пробелы, а не табуляцию.*

Элементы кода, имеющие переносы на следующую строку выравниваются с помощью висячего отступа либо вертикально, используя неявную линию в скобках (круглых, квадратных или фигурных). При использовании висячего отступа в первой линии не должно быть аргументов, как это показано в следующем примере:

```
# больше отступов для отличия заголовка
def long_function_name(
    var_one, var_two, var_three,
    var_four):
  print(var_one)
# _выровнено по открывающейся скобке
foo = long_function_name(var_one, var_two,
                         var_three, var_four)
```

Закрывающие скобки (круглые, квадратные или фигурные) в многострочных конструкциях должны ставиться под первым не пробельным символом последней строки списка или быть под первым символом строки, которая начинает конструкцию, например:

```
list1 = [
    1, 2, 3,
    4, 5, 6,
    7, 8, 9
    ]
list2 = [
    11, 12, 13,
    14, 15, 16,
    17, 18, 19
]
```

Максимальная длина строки ограничивается 79 символами, а строки документации или комментарии следует ограничивать до 72 символов. Ограничение длины строки необходимо из-за ширины окна редактора и позволяет иметь несколько открытых файлов. Для переноса длинных строк рекомендуется использовать круглые, квадратные или фигурные скобки, которые подразумевают продолжение строки в Python, поэтому длинные строки могут быть разбиты на несколько строк, обернутые в скобки, что является более предпочтительным использования обратной косой черты для продолжения строки. В ряде случаев

использование обратной косой черты является предпочтительным вариантом из-за невозможности применения неявного продолжения.

С целью выделения отдельных логически законченных участков программы используются *пустые строки*. Считается, что монолитный код является более трудным для восприятия. В то же время слишком много пустых строк делает его очень разряженным и заставляет читателя пролистывать его чаще, чем это необходимо.

Функции верхнего уровня и определения классов отделяются двумя пустыми строками. Определение метода (функции) внутри класса разделяется одной пустой строкой. Дополнительные пустые строки можно использовать для разделения различных групп похожих (однотипных) функций. Пустые строки используются также для указания логических разделов в коде программы или функции.

В процессе написания программы кодировка Python должна быть ASCII (в Python 2) или UTF-8 (Python 3). При этом соответствующие файлы не должны иметь объявления кодировки. Кодирование на основе таблицы ASCII широко применялось, применяется и, как правило, известна начинающему программисту. UTF-8 является разновидностью Unicode в 8-битном виде. Символы с кодами меньше 128 представляются одним байтом (это множество содержит латинский алфавит, простейшие знаки препинания и арабские цифры). В Unicode они повторяют ASCII, поэтому текст программы, написанный только этими символами, будет являться текстом в ASCII. Максимальная длина последовательности обозначения символа UTF-8 ограничена четырьмя байтами. Кодировка UTF-8 включает себя более двух миллионов символов. Это свидетельствует о том, что в ближайшее время задумываться о смене UTF-8 не приходится. В UTF-8 символы с кодами от 128 до 2 байтов обеспечивают кодировку кириллицы, расширенной латиницы, некоторые знаки препинания и ряд других алфавитов. Коды с 2048 до 3 байтов кодируют все современные формы письменности, сложные знаки препинания, математические и другие специальные символы. Коды с 65536 до 4 байтов обозначают музыкальные символы, смайлы, редкие китайские иероглифы и «вымершие» формы письменности. Кодирование с помощью UTF-8 имеет свои достоинства и свои недостатки.

Начиная с версии Python 3.0, в стандартной библиотеке рекомендуется в идентификаторах использовать только символы ASCII, что означает использование английских слов (где это возможно), поэтому знание английского языка при программировании на Python весьма желательно. Проектам с открытым кодом для широкой аудитории рекомендуется использовать это соглашение.

При написании кода программист может использовать стандартные библиотеки, сторонние библиотеки и модули текущего проекта. Подключение этих средств называется *импортом* (import). Каждый импорт рекомендуется обозначать на отдельной строке. Строки импорта всегда помещаются в начале файла сразу после комментариев к модулю в следующем порядке:

- импорты из стандартной библиотеки;
- импорты сторонних библиотек;
- импорты модулей текущего проекта.

Рекомендуется вставлять пустую строку между каждой группой импортов. Шаблоны импортов (from import *) рекомендуется избегать, так как они делают неясным то, какие имена присутствуют в глобальном списке имен.

Пробелы в выражениях и инструкциях

Правильное использование пробелов в процессе написания программного кода делает программу удобочитаемой, что повышает ее качество. **Избегайте использование пробелов в следующих случаях**.

Непосредственно внутри скобок (круглых, квадратных, фигурных).

Правильно:

```
spam(ham[1], (eggs:  2))
```

Неправильно:

```
spam( ham[ 1 ], ( eggs:  2 ) )
```

Непосредственно перед запятой, точкой с запятой или двоеточием:

Правильно:

```
if x == 4:  print(x, y); x, y = y, x
```

Неправильно:

```
if x == 4 :  print(x, y) ; x, y = y, x
```

Сразу перед открывающей скобкой, после которой начинается список аргументов при вызове функции.

Правильно:

```
spam(1)
```

Неправильно:

```
spam (1)
```

Сразу перед отрывающей скобкой, после которой следует индекс или срез.

Правильно:

```
dict['key'] = list[index]
```

Неправильно:

```
dict ['key'] = list [index]
```

Использование более одного пробела вокруг оператора присваивания или другого оператор для того, чтобы выровнять его с другим.

Правильно:

```
x = 1
y = 2
long_variable = 3
```

Неправильно:

```
x =           1
y =           2
long_variable = 3
```

Другие рекомендации. Широкое использование бинарных операций требует их визуальной идентификации в программном коде. Рекомендуется всегда окружать эти бинарные операции пробелами с каждой стороны: операции присваивания (=, +=, -= и другие), операции сравнения (==, <, >, |=, <>, <=, >=, in, not in, is, is not), логические операции (and, or, not). Если вы используете операции с разными приоритетами, то добавляйте пробелы вокруг операций с самым низким приоритетом.

Правильно:

```
i = i+1
phi += 1
x = x*2 -1
c2 = a**2+b**2
z = (x+y) * (x-y)
```

Неправильно:

```
i=i+1
phi +=1
```

```
x = x * 2 - 1
c2 = a ** 2+b ** 2
z = (x+y) * ( x- y)
```

Не рекомендуется использовать пробелы вокруг знака =, если он используется для обозначения именованного аргумента или значения параметров по умолчанию.

Правильно:

```
def complex(real, imag=0.0):
return magic(r=real, i=imag)
```

Неправильно:

```
def complex(real, imag = 0.0):
return magic(r = real, i = imag)
```

Нельзя использовать составные инструкции (несколько команд в одной строке).

Правильно:

```
if foo == 'blah':
do_blah_thing()
do_one()
do_two()
do_three()
```

Неправильно:

```
if foo == 'blah':
do_blah_thing()
do_one(); do_two(); do_three()
```

Иногда вы можете писать тело циклов while, for или ветвь if в той же строке, если команда короткая. Рекомендуется не использовать длинные строки.

Неправильно:

```
if foo == 'blah':  do_blah_thing()
for x in lst:  total += x
while t < 10:  t = delay()
```

Надо записывать так:

```
if foo == 'blah':
do_blah_thing()
for x in lst:
total += x
while t < 10:
t = delay()
```

Негласная договоренность об именах

При написании программного кода необходимо придумывать имена некоторым объектам программы: переменным, функциям, классам, пакетам и пр. Определены основные договоренности по выбору разумных идентификаторов, что позволяет в дальнейшем идентифицировать переменную, функцию или класс. Общая рекомендация: **не используйте** одиночные буквы I, O, l (малая латинская эль) в качестве каких-либо имен, чтобы не спутать их с цифрами 0 и 1 (что зависит от применяемого шрифта). Ниже рассмотрен наиболее используемый стиль именования в коде Python и указаны случаи целесообразного использования.

Функции. Используйте слово или слова в нижнем регистре. Для удобства чтения разделяйте слова подчеркиванием. Например, **function, my_function**.

Переменные. Используйте строчную букву. Для удобства чтения разделяйте слова подчеркиванием. Например, **x, var, my_variable**.

Классы. Каждое слово начинайте с заглавной буквы. Не разделяйте слова подчеркиванием Например, **Model, MyClass.**

Методы. Используйте слово или слова в нижнем регистре. Для удобства чтения разделяйте слова подчеркиваем. Например, **class_method, method.**

Константы. Используйте одну заглавную букву, слов или слова. Для удобства чтения разделяйте слова подчеркиваем. Например, **CONSTANT, MY_CONSTANT, MY_LONG_CONSTANT.**

Модули. Используйте короткие слова или слова в нижнем регистре. Для удобства чтения разделяйте слова подчеркиванием Например, **module.py, my_module.py.**

Пакеты. Используйте короткие слова или слова в нижнем регистре Не разделяйте слова подчеркиваем. Например, **package, mypackage**.

Имена следует выбирать тщательно, чтобы обозначения, используемые в математической постановке задачи, максимально соответствовали выбранным именам.

Комментарии

Считается, что «если реализацию трудно объяснить, то это была плохая идея». Комментарии, которые противоречат логике программного кода, значительно хуже, чем отсутствие комментариев. Если изменяется программный код, то комментарии подлежат исправлению. Комментарии могут быть в форме фразы или

предложения и начинаться должны с большой буквы (если это не имя переменной), старайтесь не изменять регистр переменной. Если комментарий короткий, то в конце можно опустить точку в конце предложения. В случае, когда блок комментариев состоит из нескольких предложений, то в конце предложений точка обязательна. После точки предложения ставится два пробела. Если вы владеете английским языком, то пишите комментарий по-английски. Каждая строка комментариев должна начинаться символом #. Абзацы внутри блока комментариев разделяются строкой, которая содержит один символ #. Комментарий в строке кода (встрочный комментарий) должен отделяться от инструкции минимум двумя пробелам, начинаться символом #, после которого ставится пробел.

Философия программирования

К настоящему времени существует около 8000 заявок на улучшение языка Python, но наиболее известны PEP 8 и PEP 20. Руководство PEP 20 считают философией программирования от Тима Петерса. В руководстве сформулированы некоторые общие правила написания программного кода. В Интернете можно найти эти правила на русском языке в следующем виде:

- красивое лучше уродливого;
- явное лучше неявного;
- простое лучше сложного;
- сложное лучше запутанного;
- развернутое лучше вложенного;
- разреженное лучше плотного;
- читаемость имеет значение;
- особые случаи не настолько особые, чтобы нарушать правила;
- практичность важнее безупречности;
- ошибки не должны замалчиваться, если не замалчиваются явно;
- встретив двусмысленность, отбрось искушение угадать;
- должен существовать один и, желательно, только один очевидный способ сделать что-то, хотя этот способ поначалу может быть не очевиден;
- сейчас лучше, чем никогда;
- никогда часто лучше, чем «прямо» сейчас;
- если реализацию сложно объяснить, то идея точно плохая;

- если реализацию легко объяснить, то, возможно, идея хорошая;
- пространства имен — отличная штука! Будем использовать их чаще!

Благодаря философии PEP 8 и PEP 20 в настоящее время код программ на Python один из самых читабельных и простых. Используя перечисленные правила можно написать красивый и качественный программный код.

8 Наиболее часто реализуемые действия в среде PyCharm

Назначение меню Help/Find Action

Функция помогает быстро найти и применить любое из его действий, не копаясь в меню. Под действием мы понимаем исполняемую команду, доступную для текущего положения каретки, или любую глобальную команду, например команду, открывающую окно инструмента. Чтобы выбрать функцию, нажмите **Help | Find Action** или Ctrl+Shift+A. Если нужное действие появилось в списке, его можно выполнить, щелкнув по нему или выбрав его с помощью клавиш со стрелками и нажав Enter.

Установка Python

Скачивать Python будем с официального сайта. Не рекомендуется скачивать интерпретатор Python с других сайтов или через торрент, в них могут быть вирусы. Программа бесплатная. Заходим на

https://python.org/downloads/windows/

выбираем "latest python release" и python 3 (рис. 8.1).

Появляется страница (рис. 8.2) с описанием данной версии Python (на английском). Если интересно — можете почитать. Затем крутим в самый низ страницы, а затем открываем "download page".

Вы увидите список файлов (рис. 8.3), которые можно загрузить. Нам нужен Windows x86 MSI installer (если система 32-битная) или Windows x86-64 MSI installer (если система 64-битная). Больше из файлов нам ничего не нужно.

Ждём, пока python загрузится. Затем открываем загрузившийся файл (рис. 8.4).

Выбираем папку для установки. Вы можете выбрать любую папку на своем диске (рис. 8.5).

Выбираем компоненты, которые будут установлены. Оставьте компоненты по умолчанию, если не уверены (рис. 8.6).

Далее ждем установки Python.



Рис. 8.1. Официальный сайт Python

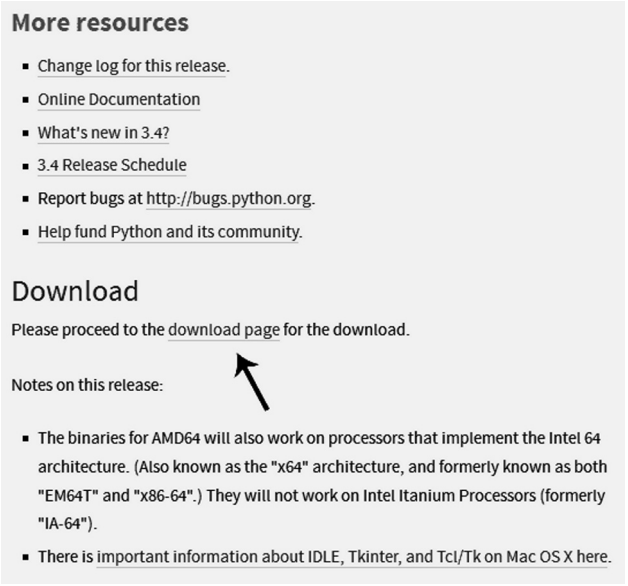


Рис. 8.2. Официальный сайт Python

Version	Operating System	Description	Date	MD5 Sum	File Size
Mac OS X 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later		316a2f83edff73bbbcb2c84390bee2db	22776248
Mac OS X 32-bit i386/PPC installer	Mac OS X	for Mac OS X 10.5 and later		534f8ec2f5ad5539f9165b3125b5e959	22692757
XZ compressed source tarball	Source release			6cafc183b4106476dd73d5738d7f616a	14125788
Gzipped source tarball	Source release			26695450087f8587b26d0b6a63844af5	19113124
Windows debug information files	Windows			9ce29e8356cf13f88e41f7595c2d7399	36744364
Windows x86 MSI installer	Windows			4940c3fad01ffa2ca7f9cc43a005b89a	24408064
Windows debug information files for 64-bit binaries	Windows			44a2d4d3c62a147f5a9f733b030490d1	24129218
Windows help file	Windows			6ff47ff938b15d2900f3c7311ab629e5	7297786
Windows x86-64 MSI installer	Windows	for AMD64/EM64T/x64, not Itanium processors		25440653f27ee1597fd6b3e15eee155f	25104384

Рис. 8.3. Список файлов



Рис. 8.4. Установка Python

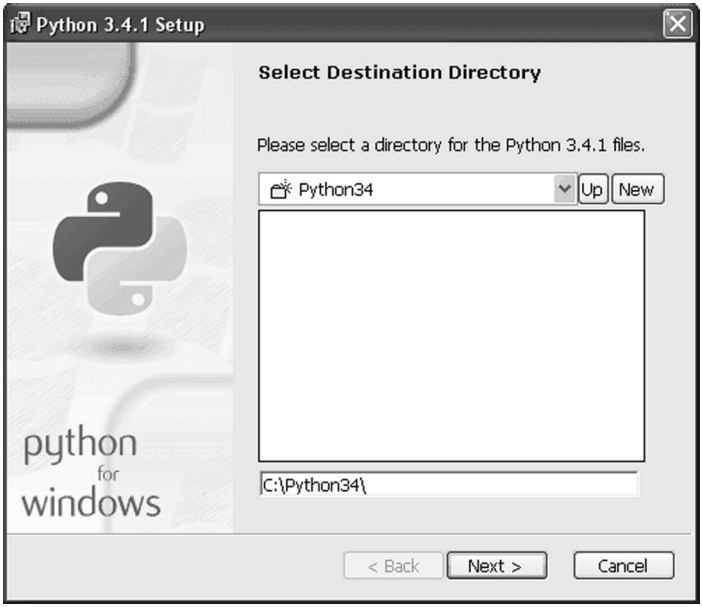


Рис. 8.5. Установка Python

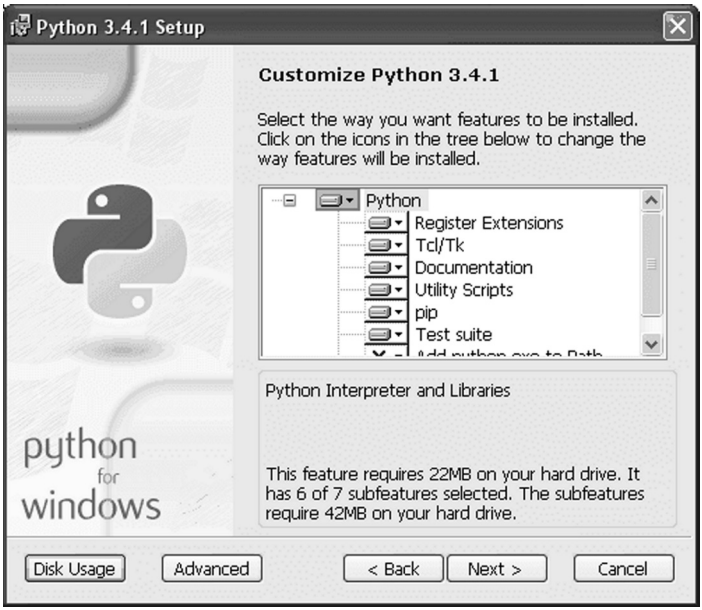


Рис. 8.6. Установка Python

Импорт/добавление существующего файл Python в проект PyCharm

Скопируйте файлы в какой-либо каталог в корне проекта с помощью вашего любимого файлового менеджера или добавьте каталог, содержащий ваши файлы, в проект, используя Settings (Preferences на Mac) | Project Structure | Добавить корневой каталог контента.

Quick фиксы (действия по быстрому исправлению кода) в PyCharm

Быстрые исправления визуально отображаются в виде индикаторов действий слева от выделенной проблемы кода. Эти индикаторы появляются при наведении курсора на проблему кода. Следующие индикаторы означают быстрые исправления.


Предлагает быстрое исправление обнаруженной проблемы кода с уровнем серьезности Error.


Предлагает рефакторинг на месте или рефакторинг, который может исправить обнаруженную проблему кода.


Предлагает быстрое исправление обнаруженной проблемы кода с уровнем серьезности Предупреждение, Предложение или Подсказка.

Навигация с помощью курсора

Для навигации назад нажмите Ctrl+Alt+влево. Для навигации вперед нажмите Ctrl+Alt+Вправо.

Чтобы перейти к последнему отредактированному месту, нажмите Ctrl+Shift+Backspace.

Чтобы найти текущее местоположение курсора в редакторе, нажмите Ctrl+M. Это действие может быть полезно, если вы не хотите прокручивать большой файл.

Однако для достижения того же результата можно нажать клавиши со стрелками вверх и вниз.

Чтобы выделить слово на месте курсора, которое вы пытаетесь найти, выберите в главном меню команду Правка | Найти | Следующее вхождение слова на курсоре (Edit | Find | Next Occurrence of the Word at Caret from the main menu).

Чтобы посмотреть, на каком элементе находится курсор, нажмите Alt+Q.

Чтобы переместить курсор между соответствующими скобками блока кода, нажмите Ctrl+Shift+M.

Чтобы перемещаться между блоками кода, нажмите Ctrl+[или Ctrl+].

Настройка собственного стиля кода

Стиль кода может быть определен для каждого языка **File/Settings/Editor/Code Style** (рис. 8.7). Вы также можете создать и сохранить свой собственный стиль кода.

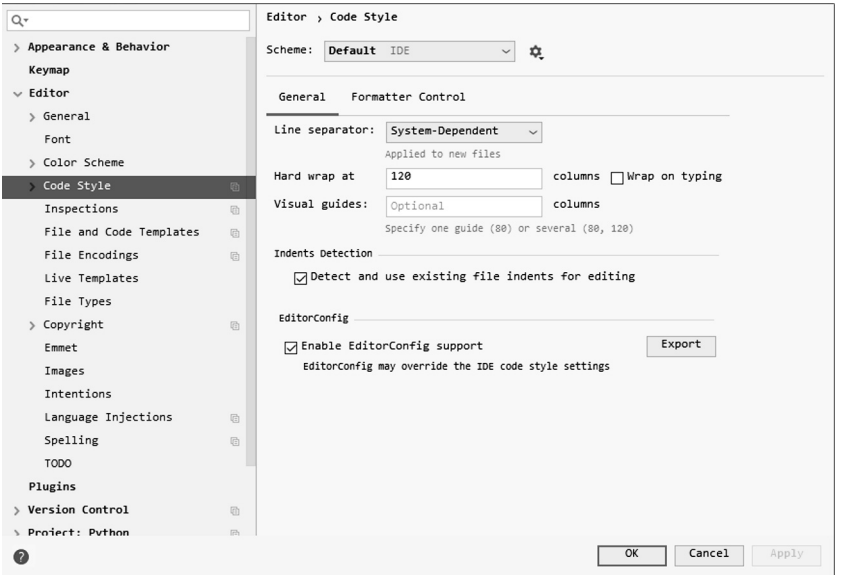


Рис. 8.7. Окно *Settings*

Настройка PyCharm + GitHub

Как создать и настроить синхронизацию репозиториев GitHub с PyCharm?

1. Для работы с системами контроля версий (например, git) в меню PyCharm смотрите на VCS.

2. Чтобы создать локальный репозиторий в проекта, вызывайте действие VCS/Import into Version Control/Create git repository.

3. Чтобы скачать удаленный репозиторий (например, из github), вызывайте действие VCS/Checkout from Control Version/Git. В появившемся диалоге указывайте ссылку на репозиторий. Возможно придется авторизоваться в Github.

4. Если у вас уже есть локальный проект и нужно на гитхаб отправить, то VCS/Import into Version Control/Share project on Github. Это создаст новый репозиторий в Github.

Просмотр структуры модуля

Окно инструмента Структура: View | Tool Windows | Structure или Alt+7. Всплывающее окно Структура: Navigate | File Structure or Ctrl+F12.

По умолчанию PyCharm показывает все классы, методы и другие элементы текущего файла. Чтобы переключить элементы, которые вы хотите показать, нажмите соответствующие кнопки на панели инструментов окна Структура.

Нажмите кнопку  «Показать поля», чтобы показать поля класса.

Нажмите кнопку  «Показать защищенные члены», чтобы в дереве отображались защищенные члены класса.

Нажмите кнопку  Show Private Members, чтобы в дереве отображались закрытые члены класса.

Нажмите кнопку  Show Inherited, чтобы показать наследуемые члены.

Нажмите кнопку  «Показать лямбды», чтобы показать лямбды.

9 Программирование на Python в Jupyter Notebook

Общие сведения о Jupyter Notebook

Для эффективной разработки программ на языке Python помимо *PyCharm*, считающейся самой интеллектуальной Python IDE с полным набором средств, часто используют *Jupyter Notebook*, который является развитием проекта IPython*. Это очень распространенное кроссплатформенное приложение, которое применяется повсеместно разработчиками на Python, а также учеными для презентации своей работы, интерактивного показа текста или фрагментов исходного кода программы на Python и результатов проведенной работы. Система *Jupyter* в отличие от IPython поддерживает более 100 языков программирования (называемых «ядрами»), включая Python, Java, R, Julia, Scheme, Scala, MatLAB и др., однако наиболее распространен вариант использования *Jupyter Notebook* для Python. Выполнить тестовый запуск полноценной версии *Jupyter* можно на сайте

https://try.jupyter.org

Jupyter Notebook — невероятно мощный инструмент для интерактивной разработки, представления данных и их анализа. В нем можно разрабатывать, документировать и выполнять приложения на языке Python. Он состоит из двух компонентов:

1) *веб-приложение*, запускаемое в браузере и позволяющее:

- редактировать и запускать код на языке Python в браузере с подсветкой синтаксиса, автоотступами и автодополнением кода;
- отображать результаты вычислений с медиапредставлением (схемы, графики);

* IPython (Interactive Python) — интерактивная оболочка для языка программирования Python, которая предоставляет расширенную интроспекцию (возможность запросить тип и структуру объекта во время выполнения программы), дополнительный командный синтаксис, подсветку кода и автоматическое дополнение.

- работать с языком разметки Markdown и LaTeX.

2) *ноутбуки* (блокноты) — файлы с расширением *ipynb*, которые объединяют повествовательный текст, визуализацию, математические уравнения, мультимедиа, исходный код, входные и выходные данные в единый документ; ноутбуки можно экспортировать в форматы *pdf* и *html*.

Таким образом, *Jupyter Notebook* представляет собой не статическую страницу, а интерактивную среду, которая позволяет писать и выполнять код на Python посредством веб-интерфейса.

Установка Jupyter Notebook

Есть несколько вариантов установки *Jupyter Notebook* на компьютер. Наиболее распространенным и рекомендуемым вариантом для новичков является установка дистрибутива *Anaconda*, в составе которого присутствует *Jupyter Notebook* и который автоматически ставится при установке дистрибутива.

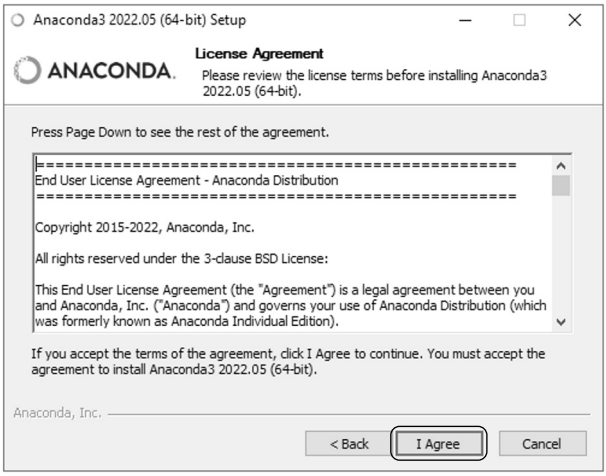
Anaconda — это широко распространенный дистрибутив для использования Python в сфере анализа данных. Его основное преимущество в том, что при его установке Python инсталлируется вместе с большинством пакетов, необходимых для обработки данных и машинного обучения (например, *Pandas* и *Numpy*), и библиотеками для создания визуализаций (рис. 9.1).



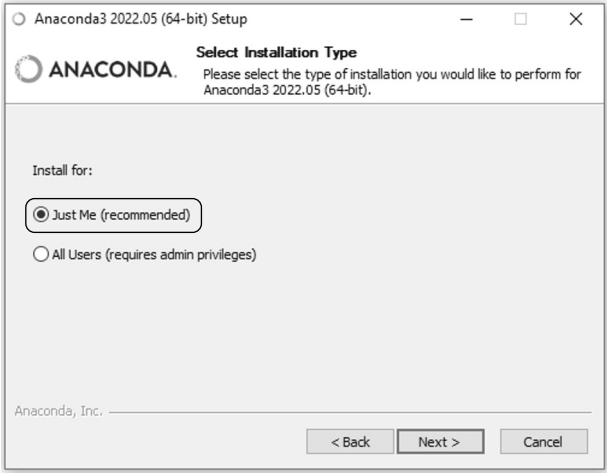
Рис. 9.1. Репозиторий дистрибутива *Anaconda*



Рис. 9.2. Кнопка для загрузки *Анаконда* с официального сайта



а)



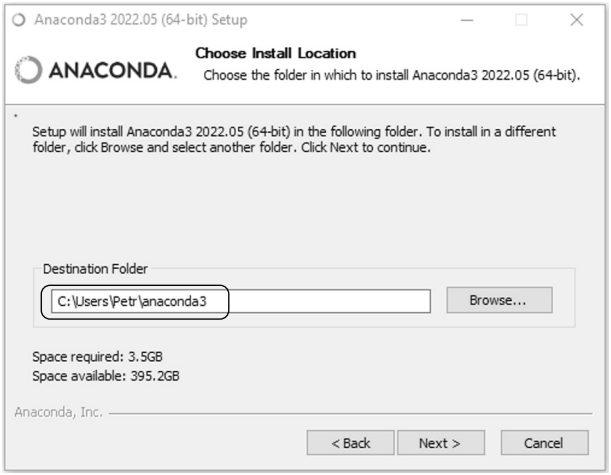
б)

Рис. 9.3. Диалоги, отображаемые при установке *Anaconda*

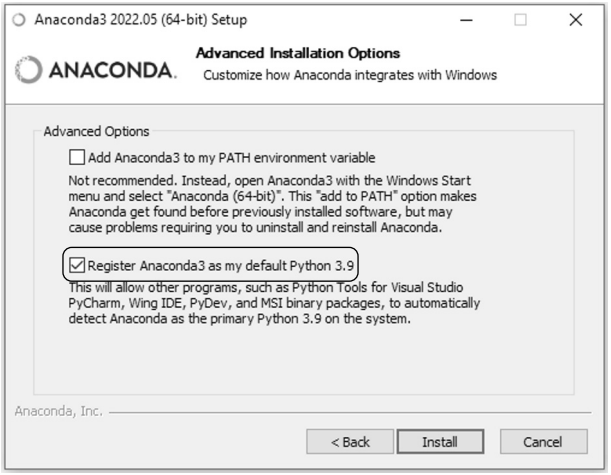
Anaconda избавляет от необходимости самостоятельно устанавливать и настраивать среду разработки для Python, а также от необходимости добавлять вручную в Python дополнительные библиотеки.

Рассмотрим процесс установки *Anaconda* для ОС *Windows*.

Шаг 1. Скачаем дистрибутив с официального сайта *Anaconda*



в)



г)

Рис. 9.3. Диалоги, отображаемые при установке *Anaconda*

https://www.anaconda.com/distribution/#windows

воспользовавшись кнопкой *Download* (Скачать).

На момент подготовки материалов предлагалась 64-битная версия с установленным Python 3.9 (рис. 9.2).

Замечание. Чаще всего на современных компьютерах установлена 64-разрядная ОС, но для уверенности перед установкой можно проверить ее разрядность в свойствах системы: → *Панель управления* → *Система*.

Шаг 2. После скачивания запускаем установочный файл и следуем инструкциям в установщике:

- принимаем лицензионное соглашение, нажав на кнопку *I Agree* (рис. 9.3,*а*);
- выбираем тип установки: для текущего пользователя (рекомендуется оставить этот вариант) или для всех пользователей (рис. 9.3,*б*);
- выбираем путь установки (рис. 9.3,*в*); здесь следует учесть, что путь к *Anaconda* должен содержать только латинские символы, поэтому если пользователь *Windows* носит русскоязычное название, то необходимо установить *Anaconda* в другую папку.

В заключительной инструкции рекомендуется:

1) НЕ добавлять *Anaconda* в переменную окружения PATH;

2) зарегистрировать *Anaconda* в качестве Python, который будет использоваться в системе по умолчанию (рис. 9.3,*г*);

3) оставляем настройки без изменения и нажимаем кнопку *Install* для запуска процесса установки.

После окончания установки в стартовом меню «Пуск» появится программная папка *Anaconda3* (*64-bit*) (рис. 9.4).

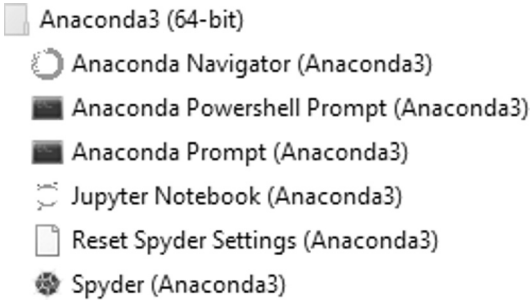


Рис. 9.4. Содержимое папки *Anaconda* стартового меню

Замечание. Если установщик выдает ошибку о том, что пользователь не имеет доступа к записи в указанную при инсталляции папку, то нужно перезапустить установщик с правами администратора.

Терминология Jupyter Notebook

Непосредственно перед знакомством с *Jupyter Notebook* необходимо разобраться с терминами, которые в нем используются

и являются ключом как к пониманию *Jupyter Notebook*, так и к тому, что делает его не просто текстовым редактором.

Markdown — это язык разметки текстов, являющийся надмножеством HTML, предназначенный для создания красиво оформленных текстов в обычных текстовых файлах (формата *txt*), а также для конвертации текстов в формат *html* или *pdf*.

Смысл использования Markdown в том, что при подготовке текста на его разметку тратят минимум усилий, а уже какой-то другой плагин или программа, поддерживающая язык разметки Markdown, превращает эту разметку в итоговый документ (например, в *html*). Markdown — это как бы язык для других программ (метаязык), передающий им правила формирования документов на основе вашего текста.

Пример. В Markdown для оформления заголовков используют решетку (#): одна решетка определяет заголовок первого уровня, две решетки — заголовок второго уровня и т. д. до пятого уровня. При подготовке текста фрагменты, соответствующие заголовкам, будут размечены следующим образом:

```
# Глава 1.  Программы с линейной структурой
- - -
## 1.1.  Линейный алгоритм
Линейный алгоритм включает последовательность команд
(указаний, действий, операторов), выполняемых друг за
другом в естественном порядке.
```

После преобразования разметки Markdown итоговый документ будет имеет вид, показанный на рис. 9.5.

Глава 1. Программы с линейной структурой

1.1. Линейный алгоритм

Линейный алгоритм включает последовательность команд (указаний, действий, операторов), выполняемых друг за другом в естественном порядке.

Рис. 9.5. Вид итогового документа

Как видно из примера, три подряд идущие знака «минус» преобразовались в горизонтальную линию.

Если при наборе теста мы просто разметим его с помощью Markdown, то он так и останется текстом, содержащим различные спецсимволы. Чтобы в результате текст, размеченный специальными символами, оказался отформатированным, используют

специальные программы (плагины), поддерживающие язык разметки Markdown. *Jupyter Notebook* поддерживает язык разметки Markdown.

Kernel (ядро) — это «вычислительный движок», который запускает и проводит внутренний анализ кода, содержащегося в ноутбуке, и контролирует все вычисления и вывод их результатов на экран. Каждый язык программирования использует отдельное ядро.

Cell (ячейка) — это контейнер для *текста*, который будет отображаться в ноутбуке, или *кода*, который будет выполняться ядром, поэтому выделяют два основных типа ячеек: Code и Markdown. Ячейки образуют структуру ноутбука.

Ячейка типа Code содержит код, который должен быть выполнен в ядре, и отображает результат его выполнения ниже.

Ячейка типа Markdown содержит текст, отформатированный с использованием Markdown, и отображает его вывод на месте при запуске.

Запуск Jupyter

Запустить *Jupyter* в *Windows* после установки *Anaconda* можно двумя способами. Вы можете выбрать любой способ из представленных.

Первый способ (самый быстрый) — запустив программу из меню **Пуск** → **Все программы** → **Anaconda3 (64-bit)** → **Jupyter Notebook (Anaconda3)**.

Второй способ — через *Anaconda Navigator*, запустив программу из меню **Пуск** → **Все программы** → **Anaconda3 (64-bit)** → **Anaconda Navigator (Anaconda3)**. После запуска появится окно программы *Anaconda Navigator* с выбором приложения. Запустить *Jupyter* можно, нажав на кнопку *Launch* на плитке приложения *Jupyter Notebook* (рис. 9.6).

Третий способ — через *Anaconda Prompt*: запустив программу из меню **Пуск** → **Все программы** → **Anaconda3 (64-bit)** → **Anaconda Prompt (Anaconda3)**. После выбора приложения отобразится окно терминала, в котором для запуска *Jupyter* нужно ввести команду jupyter notebook (рис. 9.7).

В процессе запуска любым из способов создается локальный веб-сервер *Jupyter*, прослушивающий сетевой порт с номером 8888. Автоматически в веб-браузере, используемым по умолчанию, откроется новая вкладка *Home* с оболочкой *Jupyter* (рис. 9.8) со следующим URL:

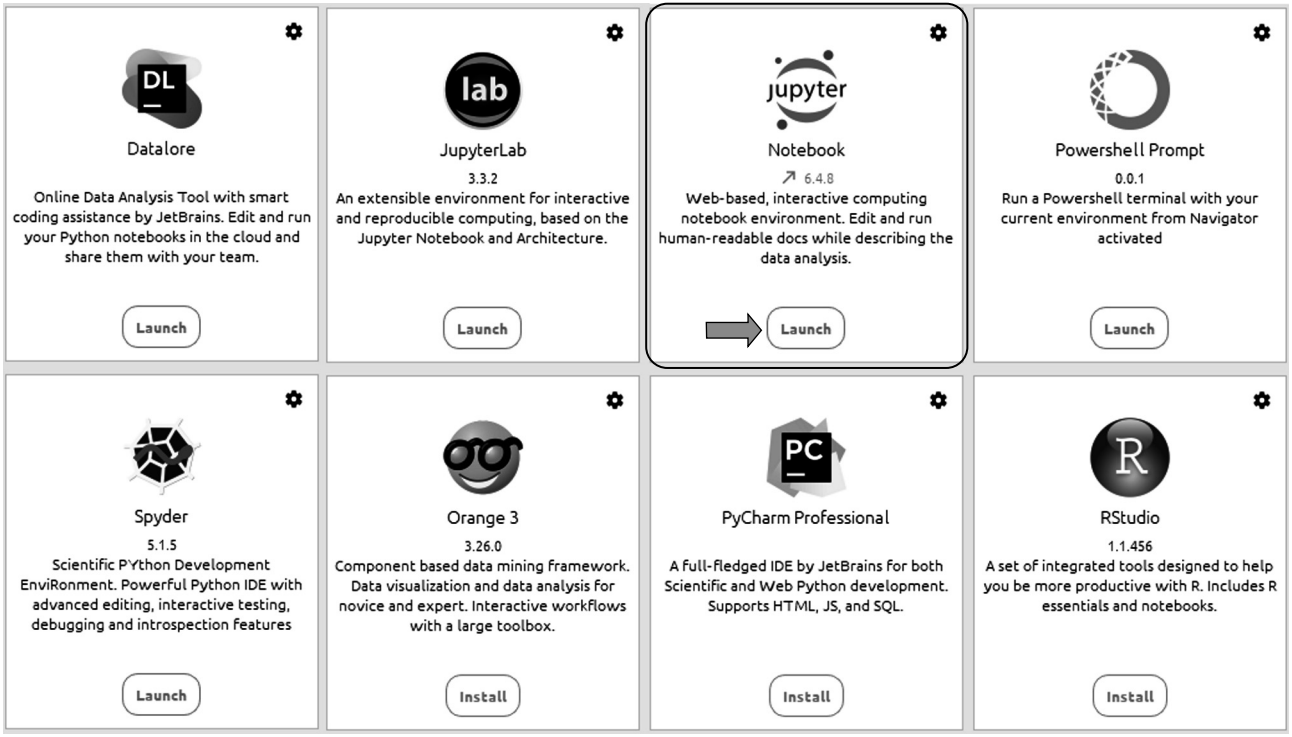


Рис. 9.6. Плитка приложений программы *Anaconda*

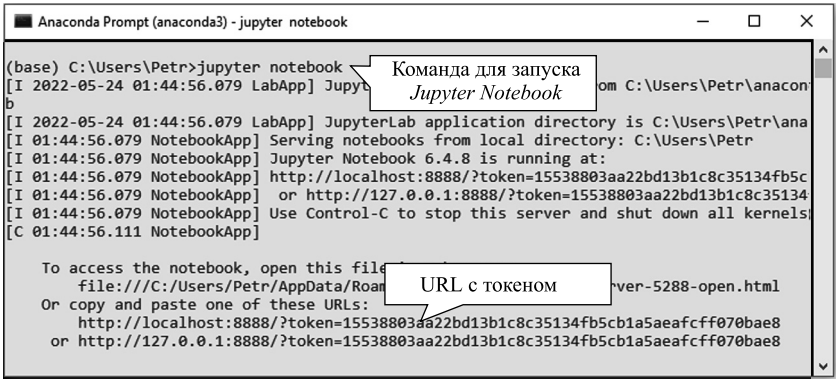


Рис. 9.7. Окно терминала программы *Anaconda Prompt*

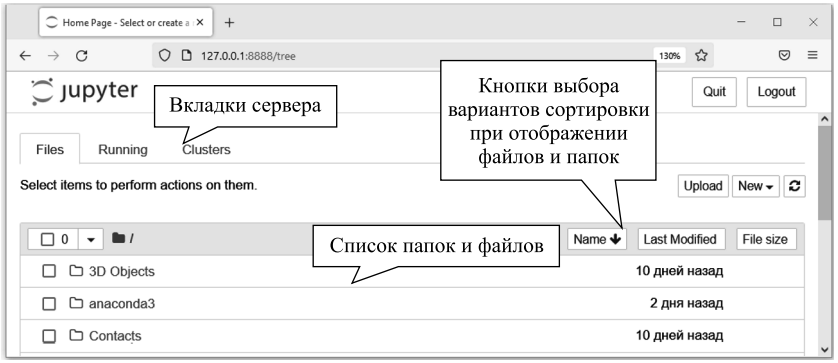


Рис. 9.8. Вид окна браузера после запуска *Jupyter Notebook*

http://localhost:8888/tree

Это *панель управления* ноутбуками *Jupyter*.

Localhost (иначе 127.0.0.1) — это адрес локального компьютера; *localhost* не является веб-сайтом, но указывает, что контент обслуживается с вашего собственного компьютера.

Панель управления предоставляет доступ только к файлам и подпапкам, содержащимся в рабочем каталоге *Jupyter*, из которого он запускается. Этот каталог определяется при установке, но может быть изменен.

Замечание. Если вы используете по умолчанию браузер, который запрещает появление всплывающих окон, то *Jupyter* может не открыться в браузере. В этом случае в окне консоли, появившемся при запуске (см. рис. 9.7), необходимо вручную скопировать строку с адресом (http://localhost:8888/tree) и токеном, после чего вставить в адресную строку браузера.

Обратите внимание! Окно консоли, появившееся после запуска локального веб-сервера (см. рис. 9.7), закрывать нельзя, иначе *Jupyter* перестанет работать. Правильно завершить работу *Jupyter* можно, дважды нажав в этом окне комбинацию Ctrl+C, а не закрыть его, нажав на крестик .

Создание и переименование папки

Для хранения своих ноутбуков создадим отдельную папку, нажав на кнопку *New* в правой части панели управления вкладки *Files* и выбрав в выпадающем списке команду *Folder* (рис. 9.9).



Рис. 9.9. Создание новой папки, файла или терминала

Новая папка получит название *Untitled Folder*. Поставив галочку напротив этой папки, можно воспользоваться кнопкой *Rename* для ее переименования (рис. 9.10). Новое имя папки вводится в появившемся окне *Переименовать директорию*.



Рис. 9.10. Переименование папки

Создание и сохранение блокнота

Для создания ноутбука — документа *Jupyter Notebook* с текстом первой программы перейдем в созданную папку (выполнив двойной клик по ней в браузере), нажмем на кнопку *New* в правой части панели управления и выберем в выпадающем списке

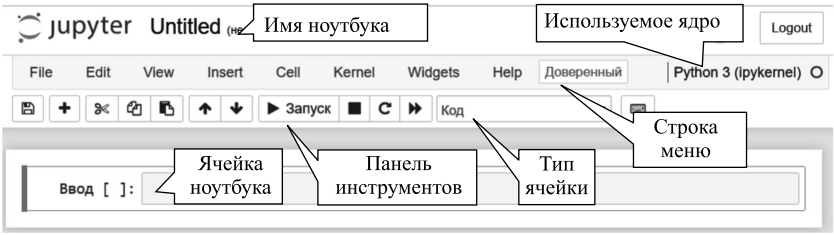


Рис. 9.11. Интерфейс ноутбука

команду *Python3* (см. рис. 9.9). В результате будет создан ноутбук, а окно браузера примет вид, показанный на рис. 9.11.

Вверху страницы отображается название ноутбука *Untitled* (заданное по умолчанию), оно отображается и в заголовке страницы. Под названием располагаются строка команд меню и панель инструментов *Jupyter Notebook* (рис. 9.11).

Переименуем ноутбук. Для этого достаточно навести на его название мышь и кликнуть по тексту. В появившемся окне *Переименовать блокнот* можно ввести новое имя (рис. 9.12) и нажать на кнопку *Переименование*.



Рис. 9.12. Диалоговое окно переименования ноутбука

Особенностью написания кода в ноутбуке является то, что код разбивается на ячейки, каждая из которых может быть выполнена отдельно.

В созданном ноутбуке по умолчанию содержится единственная ячейка типа Code (Код) и эта ячейка использует ядро, которое было выбрано при его запуске. Перед ячейкой написано Ввод [] (In []). Это ключевое слово значит, что здесь ожидается ввод кода программы. Синий контур, обрамляющий ячейку, означает командный режим. Если установить курсор внутрь ячейки, то цвет подсветки контура изменится на зеленый, это будет означать, что ячейка находится в режиме редактирования.

Ноутбук может находиться в одном из двух режимов работы с ячейками:

- *режим правки* (*Edit mode*) — когда происходит редактирование содержимого, копирование, вставка; есть возможность возвращаться к ячейкам и оперативно вносить правки;
- *командный режим* (*Command mode*) — когда происходит выполнение команд, результат выводится отдельно по каждой команде.

Текущий режим отображается в правой части строки меню (рис. 9.13). В режиме правки появляется изображение карандаша; отсутствие которого показывает, что ноутбук находится в командном режиме.



Рис. 9.13. Индикаторы режима работы с ячейками и загруженности ядра

Рядом с индикатором режима работы с ячейками находится индикатор загруженности ядра Python (см. рис. 9.13). Если ядро находится в режиме ожидания, то индикатор представляет собой окружность. Если ядро выполняет какую-то задачу, то изображение изменится на закрашенный круг. При выполнении достаточно простых операций обычно незаметно, как кружок меняет цвет. Но если Python выполняет какую-то затратную операцию (например, сложный алгоритм с большим числом итераций), индикатор достаточно долго может оставаться черным. Такую же картину можно увидеть, если Python вдруг завис.

В Jupyter Notebook можно делать все то же, что и в интерактивном интерпретаторе Python и в PyCharm. Ноутбук можно сохранять, можно вернуться к нему в дальнейшем, перезапустить ячейки — это очень удобно.

В ячейке Ввод [] блокнота *Jupyter Notebook* наберем код решения задачи нахождения первых 10 чисел последовательности Фибоначчи на языке Python (рис. 9.14).

Для запуска кода из активной ячейки воспользуемся кнопкой *Запуск* (*Run*) ▶ Запуск панели инструментов *Jupyter Notebook*.

Для удобства можно запустить ячейку на выполнение, используя следующие комбинации клавиш:

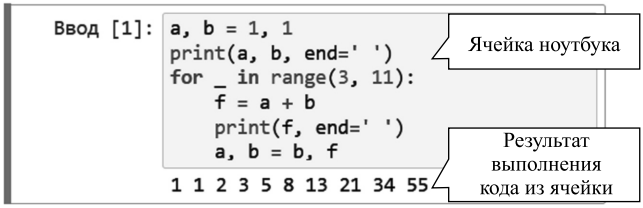


Рис. 9.14. Код программы с результатами работы

- Ctrl+Enter — запуск кода активной ячейки на выполнение (ячейка остается активной);
- Alt+Enter — запуск кода активной ячейки на выполнение и добавление новой ячейки под пекущей (активной становится добавленная ячейка);
- Shift+Enter — запуск кода активной ячейки на выполнение и переход к следующей (если под активной ячейкой нет ячеек, то добавляется новая).

Если в коде запускаемой на выполнение ячейки предусмотрен вывод результатов, то *Jupyter Notebook* отобразит вывод под ячейкой. Это очень удобно и позволяет заниматься программированием в интерактивном режиме, имея возможность отслеживать вывод каждого шага программы и тестировать каждый шаг, оперативно внося изменения при обнаружении неполадок.

Как видно из рис. 9.14, после запуска ячейки вместо Ввод [] теперь написано Ввод [1]. Появление номера (числа) в скобках строки Ввод [] означает, что код ячейки выполнен, а число N в квадратных скобках показывает номер выполненной ячейки. Последовательность номеров не строгая.

Наличие символа «звездочка» в скобках этой строки (Ввод [*]) означает, что ячейка выполняется (рис. 9.15).

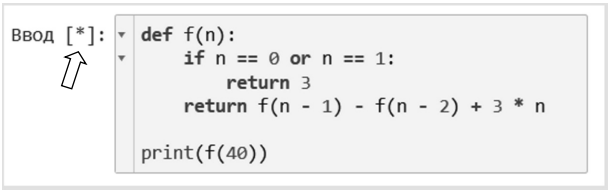


Рис. 9.15. Ячейка в режиме выполнения

Код в ячейке можно модифицировать и запускать повторно (изменится индексация ячеек), т. е. можно изменять очередность выполнения ячеек. Но при этом необходимо следить за тем, **что-**

бы интерпретатор не обращался к переменным, которые на данный момент он не знает.

На рис. 9.16, слева показан результат запуска ячейки с неопределенной переменной, при котором возникло исключение NameError. На рис. 9.16, справа показан запуск ячеек с непоследовательным порядком выполнения: первой была выполнена центральная ячейка, потом верхняя, далее нижняя. Если повторить после этих действий запуск, например, верхней ячейки, то в результат будет равен 17.

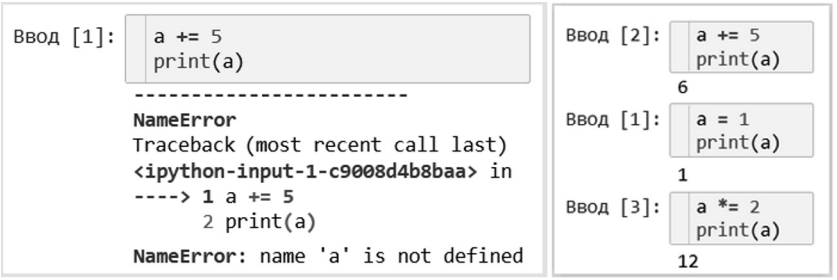


Рис. 9.16. Запуск ячейки с неопределенной переменной (слева) и с измененным порядком следования ячеек (справа)

Данный пример показывает, что *Jupyter Notebook* поддерживает нелинейную последовательность исполнения, т. е. он позволяет запускать ячейки с кодом в произвольном порядке. Конечно, это повышает требования к разработчику, который должен помнить, какая конкретно последовательность ячеек решает задачу. Но часто нелинейная последовательность исполнения оказывается удобной и востребованной, например, когда полный перезапуск решения требует нескольких часов, а надо перезапустить только определенный участок кода.

Чтобы добавить новую ячейку, можно воспользоваться кнопкой + панели инструментов. Новая ячейки будет добавлена под активной.

Jupyter Notebook поддерживает групповые операции с ячейками. Для выделения группы подряд идущих ячеек используется Shift+клик на полях слева от ячеек.

Порядок следования ячеек можно изменить с помощью кнопок ↑/↓, которые перемещают выбранные ячейки соответственно вверх/вниз.

Каждый ноутбук при открытии использует новую вкладку. При этом создается файл ноутбука с расширением *ipynb*, а также

файл контрольной точки. Файл контрольной точки будет расположен в скрытом подкаталоге с именем *.ipynb_checkpoints* в месте сохранения исходного ноутбука, он также является файлом *.ipynb*. Открытые ноутбуки автоматически сохраняются каждые 120 секунд (поэтому данные теряются довольно редко), обновляя только исходный файл *.ipynb*, а не файл контрольной точки.

При выборе команды *File* → *Save and Checkpoint* (*Файл* → *Сохранение и контрольная точка*) или комбинации Ctrl+S файлы ноутбука и контрольной точки обновляются. Следовательно, контрольная точка позволяет восстановить ноутбук в случае непредвиденной проблемы, т. е. **контрольная точка** — это резервное хранилище на случай форс-мажоров. Для сохранения ноутбука и файла контрольной точки используется также кнопка  панели инструментов.

Вернуться к контрольной точке можно через меню *File* → *Revert to Checkpoint* (*Файл* → *Вернуться к контрольной точке*). При выборе этой команды в диалоге возврата к контрольной точке необходимо подтвердить свои действия (рис. 9.17).



Рис. 9.17. Диалог возврата к контрольной точке

Если после последнего сохранения в ноутбук не были внесены изменения, а пользователь закрывает страницу браузера, то будет выведено всплывающее сообщение с подтверждением совершаемых действий (рис. 9.18).

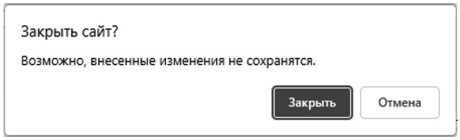


Рис. 9.18. Сообщение, возникающее при закрытии браузера

Если нужно загрузить копию файла с другим расширением, например, как *.py*, необходимо выбрать меню *File* → *Download* as → *Python* (*.py*) (*Файл* → *Загрузить как* → *Python* (*.py*)).

Интерфейс Jupyter Notebook

Горячие клавиши Jupyter Notebook

При работе в *Jupyter Notebook* всегда есть одна *активная* ячейка, выделенная рамкой, цвет которой обозначает ее текущий режим, где зеленый означает режим редактирования, а синий — командный режим. Переключение между режимом редактирования и командным режимом осуществляется с помощью кла-

Таблица 9.1

Часто используемые при работе клавиши

Сочетание клавиш	Режим работы	Назначение
Ctrl+S	Любой	Сохранить ноутбук
Tab	Редактирование	Завершение кода или отступ
Shift+Tab	Редактирование	Всплывающая подсказка
Esc	Редактирование	Вход в командный режим, после этого при помощи стрелок можно выбирать другую ячейку
Ctrl+Shift+–	Редактирование	Разделение активной ячейки по курсору
Shift+Enter	Командный	Выполнение активной ячейки + переключение фокуса на следующую существующую
Alt+Enter	Командный	Выполнение активной ячейки + создание новой ячейки снизу + переключение фокуса на нее
Ctrl+Enter	Командный	Выполнение активной ячейки, фокус остается на ней же
A	Командный	Создать пустую ячейку над активной ячейкой (от *above*)
B	Командный	Создать пустую ячейку под активной ячейкой (от *below*)
D+D	Командный	Удалить активную ячейку (нужно нажать *d* два раза)
Z	Командный	Отменить удаление ячейки
O	Командный	Скрыть/отобразить вывод активной ячейки
M	Командный	Изменить тип активной ячейки на Markdown
Y	Командный	Изменить тип активной ячейки на Code
L	Командный	Включить/выключить нумерацию строк в ячейке

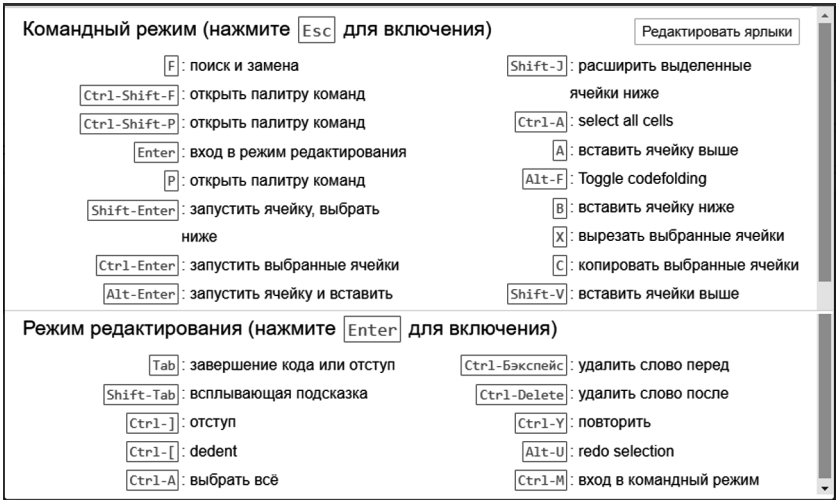


Рис. 9.19. Окно со встроенными сочетаниями клавиш

виш Esc и Enter соответственно. Важно понимать, в каком режиме находится активная ячейка, поскольку в командном режиме для ускорения рабочего процесса можно использовать сочетания клавиш, которые являются очень популярным средством *Jupyter Notebook*. В табл. 9.1 представлены часто используемые при работе в *Jupyter Notebook* клавиши (и их комбинации), их назначение.

Перечень встроенных в *Jupyter Notebook* сочетаний клавиш (рис. 9.19) доступен по нажатию комбинации Esc+H или из меню *Help* → *Keyboard Shortcuts* (*Справка* → *Горячие клавиши*).

Получение справочной информации

В *Jupyter Notebook* встроена мощная система помощи. На рис. 9.20 показана всплывающая подсказка, которая отображается при нажатии комбинации Shift+Tab во время ввода оператора print. Кнопка + раскрытия текста подсказки позволяет не только отобразить информацию по интересующему объекту языка, но и посмотреть пример его использования.

Отобразить общую информацию об объекте (интроспекция объекта) можно также с помощью ? или ??. Для функций ? показывает строку документации, ?? — по возможности показывает исходный код функции (рис. 9.21).

В *Jupyter Notebook* имеется ряд полезных команд. Отобразить командную панель можно с помощью кнопки панели инструментов со значком клавиатуры (Ctrl+Shift+P).

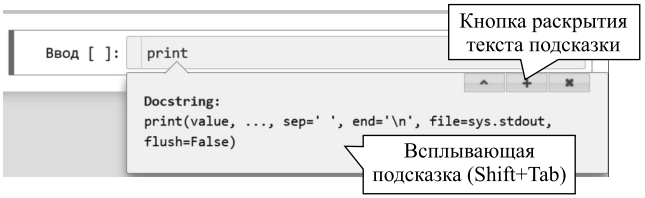


Рис. 9.20. Всплывающая подсказка

```
Ввод [1]: lst = [1, 5, -8, 6, 17, -9]
          lst?

Type:        list
String form: [1, 5, -8, 6, 17, -9]
Length:      6
Docstring:
Built-in mutable sequence.

If no argument is given, the constructor creates a new empty list.
The argument must be an iterable if specified.
```

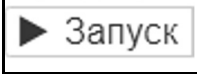
Рис. 9.21. Вывод информации об объекте

Панель инструментов

Кнопки, формирующие панель инструментов *Jupyter Notebook*, помогают выполнять часто используемые команды редактора. В табл. 9.2 приведено их назначение.

Таблица 9.2

Назначение кнопок панели инструментов

Кнопка	Назначение
	Сохранение файла ноутбука и файла контрольной точки
	Добавление новой ячейки. Новая ячейки будет добавлена под активной
	Вырезание выделенных ячеек (содержимое ячеек помещается в буфер обмена, сама ячейка удаляется)
	Копирование выделенных ячеек (содержимое ячеек помещается в буфер обмена, сама ячейка остается)
	Вставка ячейки из буфера ниже выделенной ячейки
	Перемещение выбранных ячеек соответственно вверх или вниз
	Запуск кода из активной ячейки на выполнение с последующим переходом к следующей ячейке

Окончание табл. 9.2

Кнопка	Назначение
	Прерывание работы ядра. Если в этот момент выполнялась ячейка с кодом, то выбрасывается исключение KeyboardInterrupt
	Перезапуск ядра с очисткой значений всех переменных, которые были определены в текущем сеансе работы
	Перезапуск ядра с последующим переходом к последней ячейке
	Выбор типа ячейки: Код, Markdown, Необработанный NBConvert и Заголовок
	Отображение командной панели с перечнем доступных команд и горячими комбинациями их вызова

Замечание. Технически существует четыре типа ячеек: Код, Markdown, Необработанный NBConvert и Заголовок. Но тип ячейки Заголовок больше не поддерживается, и в случае его выбора отобразится диалоговое окно с соответствующим сообщением и предложением использовать вместо этого Markdown для своих заголовков. Основные используемые типы ячеек — это типы ячеек Code и Markdown. Тип ячеек Необработанный NBConvert предназначен для написания неразмеченного текста и для особых случаев использования nbconvert инструмента командной строки, который предназначен для преобразования или экспорта вашего ноутбука в другой формат (html, pdf, LaTeX и др.).

Строка меню Jupyter Notebook

Строка меню *Jupyter Notebook* (рис. 9.22) содержит команды, обеспечивающие взаимодействие с ноутбуком, доступ к документации популярных библиотек Python, экспорта проекта для последующей демонстрации и пр.



Рис. 9.22. Строка меню

Опишем основное назначение пунктов строки меню.

В меню *File* (*Файл*) расположены команды для создания нового блокнота, переименования и открытия уже существующего, его экспорта в один из доступных форматов файлов. Здесь же

создаются контрольные точки, к которым можно откатиться в случае возникновения нештатных ситуаций.

Команды меню *Edit* (*Правка*) предназначены для выполнения операций с ячейками. С их помощью можно вырезать, копировать, вставлять, удалять, разделять или объединять ячейки, вставлять в ячейку изображение, а также изменять порядок следования ячеек в ноутбуке.

Замечание. Некоторые элементы в этом меню могут оказаться недоступными (пункт отображается серым цветом). Причина в том, что недоступные команды не применяются к типу выбранной ячейки. Например, в ячейку типа Code нельзя вставить изображение, а в ячейку типа Markdown можно. Если вы видите пункт меню серого цвета, попробуйте изменить тип ячейки и посмотрите, станет ли этот элемент доступным для использования.

В меню *View* (*Вид*) расположены команды для переключения видимости строки заголовка и панели инструментов, а также нумерации строк в ячейках.

Меню *Insert* (*Вставка*) предназначено только для вставки ячеек выше или ниже выбранной ячейки.

Команды меню *Cell* (*Ячейка*) позволяет запускать одну ячейку, группу ячеек, сразу все ячейки, можно запускать ячейки в определенном порядке. Здесь можно менять тип ячеек. Другая удобная функция этого меню — очистка вывода текущей ячейки и вывода всех ячеек ноутбука.

Используя команды меню *Kernel* (*Ядро*), можно запускать, прерывать, перезапускать или выключать ядро, а также запускать новое ядро. Когда вы запускаете ячейку кода, этот код выполняется в ядре и любой вывод возвращается обратно в ячейку для отображения. Состояние ядра сохраняется во времени и между ячейками — оно относится к документу в целом, а не к отдельным ячейкам.

Вероятно, вы не будете так часто работать с ядром, но бывают случаи, когда вы отлаживаете ноутбук и вам могут пригодиться следующие команды меню *Kernel*:

- *Restart* — перезапускает ядро, очищая все переменные (и прочие объекты), которые были определены;
- *Restart & Clear Output* — то же, что и *Restart*, но дополнительно стирает вывод, отображаемый под ячейками кода;
- *Restart & Run All* — то же, что и *Restart*, но дополнительно запускает все ячейки от первой до последней на выполнение;

- *Interrupt* — прерывает выполнение ячейки, бывает полезно, если ваше ядро зависло в вычислении, и его требуется остановить.

Меню *Widgets* (*Виджеты*) содержит команды для сохранения и очистки состояния виджета (см. далее).

С помощью команд меню *Help* (*Справка*) можно ознакомиться с пользовательским интерфейсом *Jupyter Notebook*, узнать о сочетаниях клавиш, получить справочные материалы по популярным библиотекам Python.

Приемы документирования в Jupyter Notebook

Jupyter Notebook поддерживает язык разметки Markdown, являющийся надмножеством HTML, в частности, поэтому при наборе текста несколько подряд идущих пробелов считаются за один пробел.

При написании размеченного текста изменяем тип ячейки на Markdown, вводим текст с использованием разметки Markdown и компилируем, используя комбинацию Shift+Enter.

Приемы форматирования текста

При наборе текста в ячейке типа Markdown для его стилизации можно использовать следующие приемы:

- заголовки создают с помощью символа «#» и следующего за ним пробела; один такой символ сделает крупный заголовок верхнего уровня, два — заголовок поменьше и т. д. до шестого уровня;
- символ звездочка «*» с двух сторон текста нужен для оформления этого текста курсивом;
- два символа звездочка «**» с двух сторон текста позволяют придать тексту полужирное начертание;
- для создания маркированного списка каждый его элемент начинают с символов «*», «–» или «+» и пробела;
- нумерованные списки начинают с цифры и точки, причём цифры могут быть любыми и идти не по порядку.

В табл. 9.3 представлена разметка Markdown, используемая при форматировании текста.

Как видно из табл. 9.3, ячейку типа Markdown можно использовать, если требуется вставить фрагмент программного кода, который не предусматривает дальнейший запуск пользователем. В таком случае код заключают в обратные апострофы. Если требуется вставить блок кода, можно использовать тройные

Таблица 9.3

Коды форматирования текста в разметке Markdown

Отображаемый текст	Код разметки	Примечание
Заголовок 1	# Заголовок 1	Заголовки должны начинаться с новой строки
Заголовок 2	## Заголовок 2	Заголовки занимают очень много места в окне. Самый маленький заголовок — 6 символов «#»
Курсив	*Курсив* _Курсив_	Символ «_» можно заменить на «*»
Жирный	**Жирный**	
Жирный	***Жирный курсив***	
Не курсив	_Не курсив_	Если надо написать символ, используемый в Markdown, то используйте экранирование символов — поставьте перед ним обратный слэш «\»
#Не заголовок#	\#Не заголовок\#	
Не жирный	**Не жирный**	
Цитата	> Цитата	Символ «>» используется для многоуровневого цитирования
	– – –	Введите на новой строке три (или более) нижних подчёркивания «___» или три звездочки «***», перейдите на новую строку
code	‘code‘	Символ «‘» используется для оформления программного кода
• первый элемент; • второй элемент.	- первый элемент; - второй элемент.	Начните строку с «+», «*» или «-» и пробела
1. Первый элемент. 2. Второй элемент.	1. Первый элемент. 2. Второй элемент.	Начните строку с «1.» и пробела

обратные апострофы, а также указать язык программирования, как показано на рис. 9.23, слева.

На рис. 9.23, слева показаны приемы форматирования текста в разметке Markdown, справа — отображение текста этой ячейки в браузере после компиляции (после нажатия Shift+Enter).

Поскольку язык разметки Markdown является надмножеством HTML, то при подготовке текста можно использовать *html*-теги, как показано на рис. 9.23, слева.

```
# Это заголовок 1 уровня
---
## Это заголовок 2 уровня

Тут *текст курсивом*, а можно так: <i>курсив</i>.

Тут **текст жирный**, а можно так: <b>жирный</b>.

Тут ***текст полужирный курсив***.

Тут <u>подчеркнутый</u> и оператор языка: `for a in A`.

<font color="red">Нумерованный список</font>:
1. Царь
2. Царевич

<font color="#1e9752">Маркированный список</font>:
* Диван
* Чемодан

Это фрагмент программного кода на Python:
```python
s = 'AB' * 52
while ('BB' in s) or ('AB' in s):
    s = s.replace('AA', 'B', 1)
    s = s.replace('BB', 'A', 1)
    s = s.replace('AB', 'BA', 1)
print(s)
```
> Жизнь состоит не в том, чтобы найти себя. Жизнь
состоит в том, что создать себя. *Джордж Бернард Шоу*
```

Это заголовок 1 уровня

Это заголовок 2 уровня

Тут *текст курсивом*, а можно так: *курсив*.

Тут **текст жирный**, а можно так: **жирный**.

Тут ***текст полужирный курсив***.

Тут подчеркнутый и оператор языка: `for a in A`.

Нумерованный список:

1. Царь
2. Царевич

Маркированный список:

- Диван
- Чемодан

Это фрагмент программного кода на Python:

```python
s = 'AB' * 52
while ('BB' in s) or ('AB' in s):
    s = s.replace('AA', 'B', 1)
    s = s.replace('BB', 'A', 1)
    s = s.replace('AB', 'BA', 1)
print(s)
```

> Жизнь состоит не в том, чтобы найти себя. Жизнь состоит в том, что создать себя. *Джордж Бернард Шоу*

Рис. 9.23. Приемы форматирования текста (слева) и его отображение (справа) в Markdown

Добавление таблиц

Разметка Markdown позволяет вставлять таблицы. Для разметки таблицы используются всего два символа: вертикальная черта «|» и дефис «-». Дефисы работают примерно так же, как в случае с горизонтальной линией: отделяют заголовки от других строк, при этом количество символов значения не имеет. Вертикальная черта служит границей между столбцами. На рис. 9.24 приведен пример разметки таблицы и ее отображение в браузере после компиляции Markdown-ячейки.

```
| № п/п | Фамилия И.О. | Оценка |
|-|-|-|
| 1 |Иванов И.И.      | 5 |
| 2 |Школин А.А.      | 5 |
| 3 |Назаров Р.Ю.     | 4 |
| 4 |Грирорьева Л.Д.  | 3 |
| 5 |Наумов Н.П.      | 4 |
```

№ п/п	Фамилия И.О.	Оценка
1	Иванов И.И.	5
2	Школин А.А.	5
3	Назаров Р.Ю.	4
4	Грирорьева Л.Д.	3
5	Наумов Н.П.	4

Рис. 9.24. Разметка для таблицы (слева) и ее отображение (справа)

Вставка ссылок на Интернет-ресурсы

Есть несколько способов добавления в ноутбук ссылок на внешние Интернет-ресурсы, в зависимости необходимости отображения адреса самого ресурса и заголовка, который будет отображаться рядом с курсором при наведении на ссылку.

Чтобы добавить гиперссылку, нужно добавить текст ссылки, заключив его в квадратные скобки и рядом в круглых скобках поместить URL ресурса и при необходимости текст заголовка (между квадратными и круглыми скобками не должно быть пробела):

```
[Текст ссылки](URL «Необязательный заголовок»)
```

На рис. 9.25 приведен пример добавления ссылок в ноутбук.

```
Посетите [наш сайт](http://www.rsreu.ru/ "РГРТУ")

Посетите [наш сайт](http://www.rsreu.ru/)

Наш сайт: <http://www.rsreu.ru/>
```

Посетите наш сайт

Посетите наш сайт РГРТУ

Наш сайт: http://www.rsreu.ru/

Рис. 9.25. Добавление гиперссылок в ноутбук (слева) и их отображение (справа)

Вставка медиаконтента

Самый простой способ вставки в ячейку типа Markdown картинки с локального компьютера — воспользоваться меню *Edit → Insert Image* (*Правка → Вставить изображение*), указав в диалогах путь к файлу с изображением.

Синтаксис Markdown для вставки картинок очень похож на разметку, используемую для вставки ссылок. Разница в восклицательном знаке перед первыми квадратными скобками:

```
![Alt-текст изображения](Источник изображения «Необязательный заголовок изображения»)
```

В качестве источника изображения необходимо указать локальный путь к файлу с изображением, который может быть абсолютным или относительным; здесь также можно указать URL изображения, например:

```
![Рыжий кот](Images/Image_1.jpg)
![Фото](http://www.rsreu.ru/modules/mod_imgslide/
            images/img1.jpg)
```

При вставке локального и сетевого изображения можно также использовать *html*-тег <img src="...">, где в атрибуте src указать путь к изображению, например:

```
<img src="./image.png", width=320,
            heigth=240>
<img
    src="http://www.rsreu.ru/modules/mod_imgslide/
    images/img1.jpg" height="1000" width="800">
```

Html-теги можно использовать и для вставки в ноутбук видеофайла, например можно использовать тег <video controls src="...">:

```
<video controls src="Video/video.mp4" width="600"
                        height="600">
```

В рассмотренном фрагменте кода видеофайл лежит в папке *Video*, в той же директории, что и ноутбук (сам файл с расширением *.ipynb*).

Работа с формулами

Формулы, вставляемые в ячейки типа Markdown, должны быть подготовлены в соответствии с синтаксисом LaTeX.

LaTeX — система набора и верстки текстов с формулами, стандарт для обмена и публикации научных документов.

Формулу можно вставлять внутрь текста в строке или располагать на отдельной строке, по центру и с номером. Поэтому выделяют несколько режимов вставки формулы в ноутбук:

- *встроенный режим* — для вставки формулы использует синтаксис: $ formula $; в этом режиме при отображении формула будет смешана с текстом до и после нее;
- *блочный режим* — для вставки формулы использует синтаксис: $$ formula $$; в этом режиме при отображении формула выделяется в отдельную строку, выравнивается по центру и отделяется сверху и снизу пустыми строками;
- для добавления длинных и сложных математических выражений часто используется *окружение* equation:

  \begin{equation}\label{ссылка} ... \end{equation},

которое в LaTeX позволяет организовать автоматическую нумерацию формул и ссылок на них, но для использования этой возможности в *Jupyter Notebook* потребуется установить соответствующее расширение.

Формулами считаются как целые формулы, так и отдельные цифры или буквы, в том числе буквы греческого алфавита, а также верхние и нижние индексы и спецзнаки. Один пробел внутри исходного текста, задающего формулу, игнорируется.

При подготовке формул используются следующие **специальные символы**, которые не выводятся на экран:

{ — начало группы;
} — конец группы;
_ — нижние индексы;
^ — верхние индексы;
\ — сигнальный символ команд.

Если эти символы должны сами присутствовать в формуле, то их следует экранировать и они будут отрисованы в формуле, например

$x \{j \}$.

Рассмотрим пример набора формул, в котором показаны встроенный и блочный режимы вставки математических выражений. На рис. 9.26 приведен пример разметки, а на рис. 9.27 — отображение текста с формулой в браузере после компиляции ячейки.

На рис. 9.28, справа приведен пример использования окружения

\begin{equation}...end {equation}

```
Функция распознавания может быть записана в виде

$$f(x,W)=h^{(2)}\left(\sum\limits_{i=1}^D
w_i^{(2)}h^{(1)}\left(\sum\limits_{j=1}^n
w_{ji}^{(1)}x_j+b_i^{(1)}\right)+b^{(2)}\right)$$

Здесь  $x$ – исходный объект (описанный $11$ признаками); $x_j$ –
соответствующий признак; $n$ –  количество нейронов во входном
слое сети, совпадающее с количеством признаков; $D$ –  количество
нейронов в скрытом слое сети; $w_i^{(2)}, w_{ji}^{(1)},
b_i^{(1)}, b^{(2)}$ – параметры сети, соответствующие весам
нейронов; $h^{(1)}, h^{(2)}$ – функции активации.
```

Рис. 9.26. Использование формата LaTeX при наборе формул

Функция распознавания может быть записана в виде

$$f(x, W) = h^{(2)}\left(\sum_{i=1}^{D} w_i^{(2)} h^{(1)}\left(\sum_{j=1}^{n} w_{ji}^{(1)} x_j + b_i^{(1)}\right) + b^{(2)}\right)$$

Здесь x – исходный объект (описанный 11 признаками); x_j – соответствующий признак; n – количество нейронов во входном слое сети, совпадающее с количеством признаков; D – количество нейронов в скрытом слое сети; $w_i^{(2)}, w_{ji}^{(1)}, b_i^{(1)}, b^{(2)}$ – параметры сети, соответствующие весам нейронов; $h^{(1)}, h^{(2)}$ – функции активации.

Рис. 9.27. Отображение формул, подготовленных в LaTeX

при наборе формул в ячейке типа Markdown, а на рис. 9.28, слева — отображение набранных формул после компиляции ячейки.

```
**Пример 1.**
\begin{equation}
  y = x_0 + \cfrac{1}{x_1
          + \cfrac{1}{x_2
          + \cfrac{1}{x_3 + \cfrac{1}{x_4} } } }
\end{equation}

**Пример 2.**
\begin{equation}
    A_{n,m} =
          \begin{pmatrix}
               a_{1,1} & a_{1,2} & \cdots & a_{1,m} \\
               a_{2,1} & a_{2,2} & \cdots & a_{2,m} \\
               \vdots  & \vdots  & \ddots & \vdots  \\
           a_{n,1} & a_{n,2} & \cdots & a_{n,m}
          \end{pmatrix}
\end{equation}
```

Пример 1.

$$y = x_0 + \cfrac{1}{x_1 + \cfrac{1}{x_2 + \cfrac{1}{x_3 + \cfrac{1}{x_4}}}}$$

Пример 2.

$$A_{n,m} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,m} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,m} \end{pmatrix}$$

Рис. 9.28. Запись и отображение формул в LaTeX

В табл. 9.4 представлены часто используемые математические символы и правила их оформления в LaTeX.

Таблица 9.4

Список математических символов

Строчные и прописные греческие буквы α \alpha β \beta ε \epsilon λ \lambda η \eta μ \mu π \pi σ \sigma Γ \Gamma Δ \Delta Θ \Theta Σ \Sigma Φ \Phi Ω \Omega
Символы бинарных отношений $\leqslant$ \leq $\geqslant$ \geq $\in$ \in $\notin$ \notin $\subset$ \subset $\subseteq$ \subseteq $\approx$ \approx $\neq$ \neq
Символы бинарных операций $\pm$ \pm $\cdot$ \cdot $\div$ \div $\times$ \times $\cup$ \cup $\cap$ \cap $\vee$ \vee $\wedge$ \wedge
Символы стрелок $\rightarrow$ \rightarrow $\leftrightarrow$ \leftrightarrow $\Leftrightarrow$ \Leftrightarrow $\uparrow$ \uparrow $\downarrow$ \downarrow $\updownarrow$ \updownarrow $\Uparrow$ \Uparrow $\Downarrow$ \Downarrow $\Updownarrow$ \Updownarrow
Прочие символы ∞ \infty $\Re$ \Re $\Im$ \Im $\dots$ \dots $\cdots$ \cdots $\vdots$ \vdots $\forall$ \forall $\exists$ \exists $\neg$ \neg $\angle$ \angle $\parallel$ \parallel $\perp$ \perp
Математические алфавиты $\mathcal{ABC}$ \mathcal{ABC} $\mathscr{ABC}$ \mathscr{ABC} $\mathfrak{ABC}$ \mathfrak{ABC} $\mathbb{ABC}$ \mathbb{ABC}

Рассмотрим примеры записи формул в LaTeX и результат их отображения в браузере (табл 9.5).

Информацию по наборе формул в LaTeX можно посмотреть по ссылке

https://en.wikibooks.org/wiki/LaTeX

Веб сервис *Detexify* (http://detexify.kirelabs.org/classify.html) по нарисованному в браузере символу, который требуется вставить в LaTeX, выдает соответствующую команду для LaTeX.

Примечание. Редактор формул Microsoft Office 2007 и более новых версий также использует команды LaTeX для набора формул. Для многих символов можно найти подсказку, наведя курсор мыши на палитру символов (рис. 9.29).

Таблица 9.5

Примеры записи формул в LaTeX и их отображения

Формула	Результат
$$\frac{dy}{dx}=\frac{3x^2}{y^3}$$	$\frac{dy}{dx}=\frac{3x^2}{y^3}$
$$\arcsin(x^2_{23} y_2)$$	$\arcsin(x^2_{23} y_2)$
$$\int\limits_0^1(x^2+4)\, dx$$	$\int\limits_0^1(x^2+4)\, dx$
$$\sqrt[3]{x^2+y^2}$$	$\sqrt[3]{x^2+y^2}$
$$x \le \pi$$	$x \le \pi$
$$\log_3(x-2)=1$$	$\log_3(x-2)=1$
$$\lim_{x \to \infty}\left(1+\frac{1}{x^2}\right)^{x^2}= e$$	$\lim_{x \to \infty}\left(1+\frac{1}{x^2}\right)^{x^2}= e$
$$\sum_{n=1}^{\infty}\frac{(-1)^n}{1+n}= \ln2$$	$\sum_{n=1}^{\infty}\frac{(-1)^n}{1+n}= \ln2$

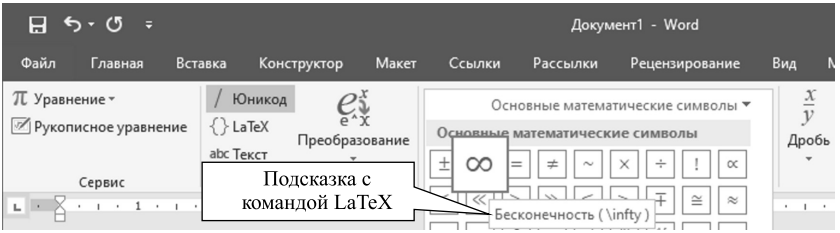


Рис. 9.29. Отображение подсказок с командами LaTeX в MS Word

Дополнительные возможности кодовых ячеек

Написание кода программы на языке Python в *Jupyter Notebook* производится в ячейки типа Code. Рассмотрим основные приемы работы с ячейками типа Code, повышающие продуктивность при использовании *Jupyter Notebook*.

Установка сторонних пакетов в Jupyter Notebook

Обычно *Jupyter*-ноутбуки начинаются с ячейки кода, содержащей необходимые импорты с помощью инструкции import, после которой указывается название подключаемого модуля (модулей). Пример импорта модулей:

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
```

Перед импортом внешнего модуля необходимо предварительно выполнить процедуру установки (инсталляции) соответствующего пакета с помощью имеющейся (начиная с версии 3.4) в комплекте поставки Python утилиты pip, которая самостоятельно найдет запрошенную библиотеку из внешнего репозитория: Github, Bitbucket, Google Code или официального PyPI (*Python Package Index*, реестра пакетов Python), загрузит дистрибутив с этой библиотекой, совместимый с версией Python, и установит ее. Если инсталлируемая библиотека требует для работы другие библиотеки, они также будут установлены.

Jupyter Notebook позволяет запустить установку внешнего пакета прямо из ячейки Code, выполнив в ячейке команду вида

```
!pip install <название пакета>
```

где <название пакета> — имя библиотеки, которую требуется установить.

Комментирование кода

Комментарии — это текстовые пояснения, которые находятся непосредственно в самой программе, улучшают понимание кода и не влияют на ход выполнения программы. Во время обработки исходного кода комментарии игнорируются компилятором и интерпретатором. Комментарии позволяют их автору быстро разбираться в ранее написанном коде, а другим разработчикам — значительно улучшить понимание текста программы во время работы над общим проектом.

В Python комментарий пишется с использованием специального символа # (решетка), после которого идет текст комментария. Комментарий, содержащийся в строке с выражением, называется *inline* комментарий.

Чтобы закомментировать сразу несколько строк программного кода (или раскомментировать их), надо:

1) выделить нужные строки;

2) нажать комбинацию Ctrl+Num </>.

Благодаря специальному синтаксису, комментарии выделяются на фоне инструкций, предназначенных для выполнения компьютером.

Вывод значений

Как правило, в ячейке отображается только последний вывод (рис. 9.30, слева). Для всего остального нужно прописывать print(). Это не всегда удобно.

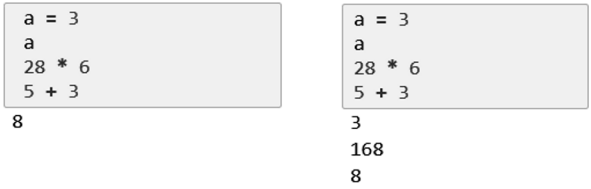


Рис. 9.30. Вывод результатов в *Jupyter Notebook* (слева — стандартный режим, справа — настроенный)

Изменить ситуацию можно, запустив в блокноте строки:

```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = 'all'
```

После выполнения этих команд *Jupyter Notebook* будет выводить результат выполнения каждой операции в отдельной строке (рис. 9.30, справа).

Для возврата к обычному представлению надо выполнить команду

```
InteractiveShell.ast_node_interactivity = 'last_expr'
```

Замечание. Запускать команду для изменения настроек необходимо в отдельной ячейке. Тогда изменения вступят в силу при следующем выполнении ячейки.

Магические команды

Одна из крайне удобных составляющих *Jupyter Notebook* — это специальные *магические команды*, облегчающие процесс разработки и расширяющие возможности. Они используют метаязык *Jupyter Magics*, команды которого обычно начинаются с % или %%. Строки и ячейки, начинающиеся с %, не интерпретируются как обычный код Python.

Магические команды могут быть двух видов:

- *line magics* — начинаются с %, выполняются для одной строки;
- *cell magics* — начинаются с %%, выполняются для всей ячейки.

Рассмотрим часто используемые магические команды и примеры их вызова.

Контроль времени выполнения. Магические команды %%time и %timeit используются для измерения времени работы кода.

%%time позволяет получить информацию о времени работы кода в рамках одной ячейки (рис. 9.31).

```
%%time
x = [(i**10) for i in range(100_000)]
```

```
Wall time: 32.9 ms
```

Рис. 9.31. Пример использования команды %%time

%timeit запускает переданный ей код по умолчанию 100000 раз и выводит информацию о среднем значении и стандартном отклонении на 7 наиболее быстрых прогонах (рис. 9.32).

```
%timeit x = [(i**10) for i in range(10)]
```

```
2.31 µs ± 23.2 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

Рис. 9.32. Пример использования команды %%timeit

Команда %timeit может принимать аргументы:

- -n<N> — количество циклов выполнений;
- -r<R> — количество повторений в каждом цикле (каждый из которых состоит из <N> повторений) с выбором наилучшего результата (по умолчанию 7);
- -p<P> — точность вывода цифр после.

Например, код ниже выполнится в цикле 2 раза, в каждом из которых он запустится по 3 раза с выбором наилучшего результата. После произойдет расчет среднего и стандартного отклонения:

```
%timeit -n 2 -r 3 x = [(i**10) for i in range(10)]
```

Импорт и экспорт кода. Магическая команда %%writefile <имя_файла.py> сохраняет содержимое (код) своей ячейки в файл <имя_файла.py>. Если указать только имя файла (а не абсолютный путь), то файл будет создан в той директории, где находится файл ноутбука.

Для загрузки кода из файла напрямую в ячейку используется команда

```
%load <имя_файла.py>.
```

Использование языков внутри блокнота. *Jupyter Notebook* позволяет модифицировать ячейку в язык с помощью соответствующей магической команды: %%python2, %%ruby, %%perl, %%html, %%latex и т. д.

Ранее рассматривался вопрос вставки в ячейку типа Markdown видеофайла. Использование магической команды %%html в ячейке типа Code позволит добавить в ноутбук файл с видео, воспользовавшись *html*-разметкой, как в примере:

```
%%html
<video width="320" heigth="240" controls>
    <source src="Media/Life.mp4" type="video/mp4">
</video>
%lsmagic
```

%lsmagic отображает встроенные магические команды.

Информацию по магическим командам можно найти по адресу

https://ipython.org/ipython-doc/3/interactive/magics.html

Интерактивное управление с помощью виджетов

Для анализа результата выполнения кода в ячейке часто приходится запускать ее несколько раз, немного меняя значения входных данных или параметров функций. Это не только непродуктивно, но часто отвлекает от основной задачи анализа данных. Для решения этой проблемы можно использовать инструмент интерактивного управления — IPywidgets. Эта библиотека позволяет с помощью виджетов менять переменные, не внося изменений в код, превращая ноутбук из статичного текстового документа в диалоговую панель, удобную для визуализации и работы с данными.

Виджеты — графические элементы, такие как кнопка, выпадающий список или текстовое поле, которые постоянно находится в браузере и позволяют нам контролировать код и данные, реагируя на события и вызов указанных обработчиков. В *Jupyter Notebook* это специальные биты кода, которые связывают воедино код Python, работающий в ядре, и JavaScript/HTML/CSS, работающий в браузере. Их прямое назначение — сделать ноутбук более динамичным, добавить в него различные элементы управления. Наиболее популярные виджеты представлены на рис. 9.33. Эти графические элементы могут быть собраны и настроены для создания сложных панелей мониторинга.

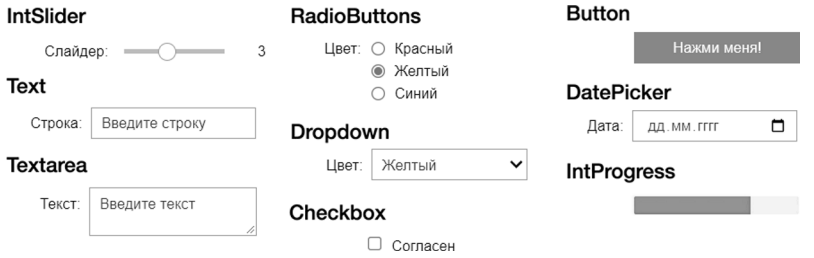


Рис. 9.33. Часто используемые виджеты

Виджеты могут выполнять код при определенных действиях, позволяя обновлять результат выполнения ячейки без повторного запуска при изменении исходных данных. На рис. 9.34 показан пример использования виджета IntSlider, который создает ползунок и отображает изменяющиеся целые значения.

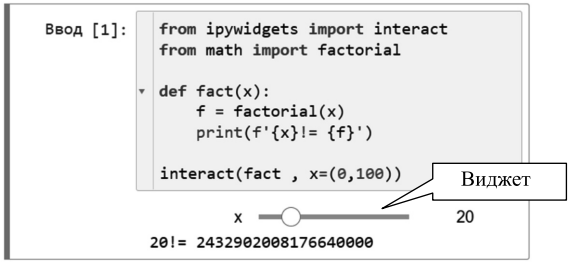


Рис. 9.34. Пример использования виджета IntSlider

Interact — самая часто используемая функция библиотеки ipywidgets, преобразующая функцию в интерактивный виджет, автоматически создающая элементы управления пользовательского интерфейса. Первый аргумент этой функции — обработчик события, а второй — значение, переданное самому обработчику события (формат: начало, конец, шаг). Она создает слайдер и вызывает некоторую функцию при его изменении. Ему можно придумать массу полезных применений: например, можно смотреть, как ваше решение зависит от изменения какого-то параметра или признака, просто двигая слайдер. Все это реализуется в одну строчку.

Если в interact передается кортеж из двух целых чисел (min, max), то создается слайдер с целочисленным значением от указанного минимума до максимума (включительно). В этом случае используется размер шага по умолчанию, равный 1. На са-

мом деле мы можем получить тот же результат, если передадим IntSlider в качестве аргумента:

```
interact(f, x=widgets.IntSlider(min=0,
            max=100, step=1, value=20))
```

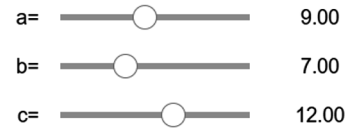
Помимо виджета IntSlider, существуют также FloatSlider (изменение вещественного числа в диапазоне) и IntRangeSlider (изменение целого числа в диапазоне). На рис. 9.35 показан пример использования виджетов FloatSlider при вычислении площади треугольника по трем сторонам по формуле Герона.

```
from ipywidgets import interactive_output
from ipywidgets.widgets import FloatSlider, VBox
from IPython.display import display

def area(a, b, c):
    p = (a + b + c) / 2
    if p < a or p < b or p < c:
        print('Треугольник не существует')
    else:
        s = (p * (p - a) * (p - b) * (p - c)) ** 0.5
        print(f'Площадь треугольника равна {s:0.2f} кв.ед.')

a = FloatSlider(min=0.5, max=20, step=0.1, value=5, description='a=')
b = FloatSlider(min=0.5, max=20, step=0.1, value=5, description='b=')
c = FloatSlider(min=0.5, max=20, step=0.1, value=5, description='c=')
dashboard = VBox([a, b, c])
out = interactive_output(area, {'a': a, 'b': b, 'c': c})

display(dashboard, out)
```




```
Площадь треугольника равна 31.30 кв.ед.
```

Рис. 9.35. Пример использования виджетов FloatSlider

Пояснение. Interactive_output обеспечивает дополнительную гибкость: вы можете контролировать расположение элементов пользовательского интерфейса. VBox используется для создания контейнера (панели приборов), в котором входные виджеты размещаются друг под другом. Функция display отображает объект виджета в ячейке ввода ноутбука.

Для получения полного списка виджетов, которыми можно украсить свой ноутбук, можно выполнить следующую команду:

```
from ipywidgets import widgets
print(dir(widgets))
```

Замечание. Примеры использования различных виджетов можно посмотреть на ресурсе

https://github.com/jupyter-widgets/ipywidgets/blob/master/docs/source/examples/Index.ipynb

Дополнительную документацию по виджетам можно найти по адресу

https://ipywidgets.readthedocs.io/en/latest/

Дополнительные настройки Jupyter Notebook

Расширения

Несмотря на то что в *Jupyter Notebooks* встроено множество функций, повышающих продуктивность его использования, можно добавлять новые с помощью *расширений*. Чтобы ими было удобно управлять, создали специальный плагин *NBextensions configurator*, задающий интерфейс для расширений.

Для установки плагина *NBextensions configurator* необходимо выполнить следующие шаги.

Шаг 1. Завершим работу *Jupyter Notebooks*.

Шаг 2. В командной строке ОС или командной строке *Anaconda Prompt* выполним последовательно четыре команды:

```
pip install
jupyter_nbextensions_configurator
jupyter_contrib_nbextensions
jupyter contrib nbextension install --user
jupyter nbextensions_configurator enable --user
```

Шаг 3. Запустим *Jupyter Notebooks*.

Шаг 4. После этого в *Jupyter* появилась вкладка *Nbextensions* (рис. 9.36). Надо переключиться на нее и убрать галочку в первой строке: *disable configuration for nbextensions without explicit compatibility* (*they may break your notebook environment, but can be useful to show for nbextension development*).

Теперь можно одним кликом включать или отключать нужное расширение, при этом ноутбук придется перезагружать.

В самом *Jupyter Notebooks* детально описано, как добавлять, включать и работать с расширениями. Эта информация вместе с GIF-файлом, демонстрирующем возможности расширения, будет доступна под списком после выбора нужного расширения.

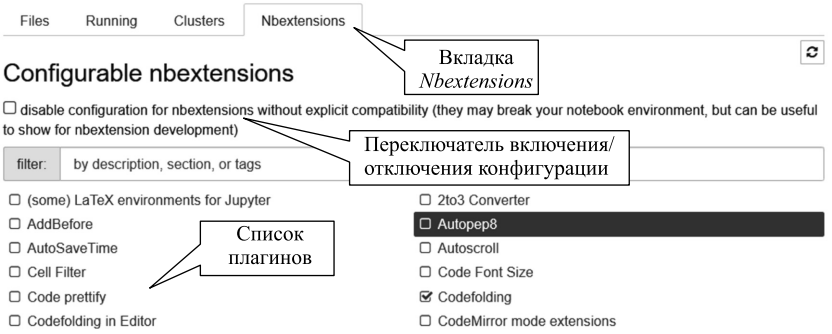


Рис. 9.36. Интерфейс *NBextensions* для управления расширениями

Таблица 9.6

Описание популярных расширений

Расширение	Описание
Hinterland	Добавляет меню автозаполнения кода в ячейке для каждого нажатия клавиши, а не только для клавиши Tab
Autopep8	Автоматически форматирует Python-код в соответствии с руководством PEP 8 по оформлению кода
Table of Contents	Собирает все выполняемые заголовки и отображает их в плавающем окне, в формате боковой панели или в меню навигации
A Code Prettifier	Подчищает, форматирует и проставляет отступы
Code Folding	В режиме редактирования добавляет иконку-треугольник для скрытия кода. Удобно при наличии длинных функций, которые для читабельности лучше скрывать
Ruler	Добавляет визуальную границу ширины кода, показывая красным границу в 80 символов в строке. Плагин нужно поставить обязательно, чтобы при конвертации ноутбука в *pdf* не столкнуться с проблемой, что что-то не влезло в одну строку
ExecuteTime	Автоматически запоминает время запуска ячейки сколько времени ушло на выполнение
LiveMarkdown-Preview	Включает режим предпросмотра для ячейки типа Markdown, при котором еще при вводе содержимого ячейки под ней будет отображаться результата ввода
Variable Inspector	Отображает информацию об используемых переменных: имя, тип, размер, значение

В табл. 9.6 представлены наиболее популярные и востребованные при подготовке ноутбуков расширения.

Замечание. Для работы расширения Autopep8 необходимо, чтобы на локальной машине (из командной строки *Anaconda Prompt*) была выполнена команда

```
pip install autopep8 --user
```

Для преобразования написанного кода в соответствии со стилевым руководством PEP8 на панели инструментов появиться иконка в виде молота .

Поскольку задачи, решаемые в *Jupyter Notebook*, разнообразны, то нет общего списка расширений, какие необходимо ставить. Его лучше всего определять на собственном опыте, индивидуально.

На рис. 9.37 показан внешний вид окна *Jupyter Notebooks* с установленными расширениями (Table of Contents, Autopep8, Variable Inspector, ExecuteTime).

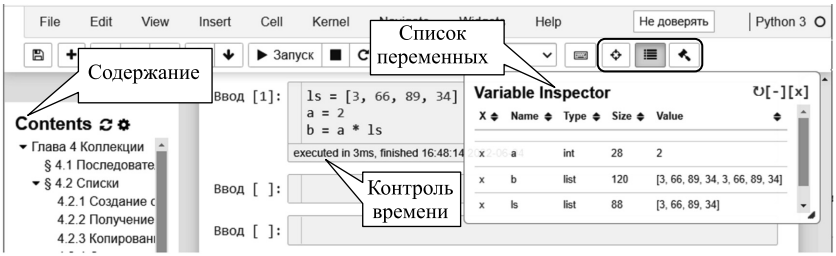


Рис. 9.37. *Jupyter Notebooks* с дополнительными расширениями

Изменение темы оформления

Помимо функциональности при работе *Jupyter Notebook* многих интересует и эстетическая сторона этого инструмента, а именно тема оформления, используемая в редакторе. Для многих программистов важно при работе использовать темную тему, которая «бережет» глаза.

Для изменения темы оформления *Jupyter Notebook* выполним следующий шаги.

Шаг 1. Установим пользовательские темы для ноутбука (из командной строки *Anaconda Prompt*), выполнив команду

```
pip install jupyterthemes
```

Шаг 2. После шага 1 можно просмотреть имена установленных пользовательских тем, выполнив команду

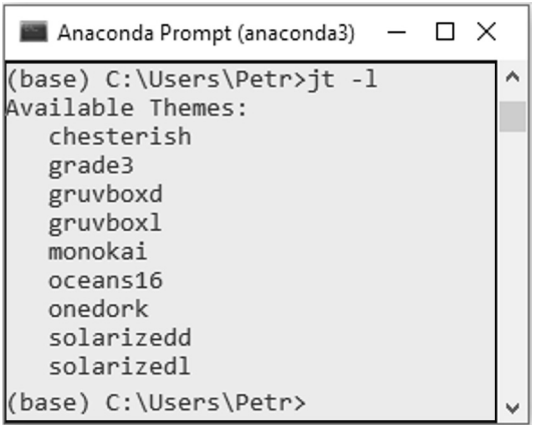


Рис. 9.38. Список тем оформления для *Jupyter Notebook*

```
jt -l
```

Названия тем отобразятся в столбец, как на рис. 9.38.

Шаг 2. Теперь можно выбирать название темы и выполнить команду `jt -t <название темы>`. Например, для изменения темы на темную:

```
jt -t onedork
```

Шаг 3. Для изменения темы в уже запущенном ноутбуке достаточно обновить браузер, воспользовавшись кнопкой .

Примечание. Если требуется восстановить исходную тему, установленную по умолчанию, введите следующий код:

```
jt -r
```

Управление папками и ноутбуками в Jupyter

На домашней странице (*Home Page*) сервера *Jupyter* после установки будут доступны три вкладки: *Files*, *Running* и *Clusters* (вкладка *NBextensions* появилась после установки соответствующего плагина).

Вкладка *Files* (рис. 9.39) предоставляет доступ к файлам и папкам в текущем каталоге, из которого было вызвано приложение *Jupyter Notebook*. Все папки в каталоге помечены иконкой с папкой , ноутбуки — иконкой записной книжки . У запущенных в работу ноутбуков будет зеленая иконка (и при этом рядом с ними написано *Running*/*Запустить*), у нерабочих — серая. Файлы и папки открываются стандартно — двойными кликом по ним в браузере. Эта вкладка содержит кнопки,

позволяющие создавать, удалять, переименовывать, скачивать, копировать файлы и директории.

С помощью кнопки *Upload* (*Загрузить*) можно загрузить файл на сервер ноутбука (рис. 9.39). Ее удобно использовать при загрузке, например, файлов с данными для анализа.

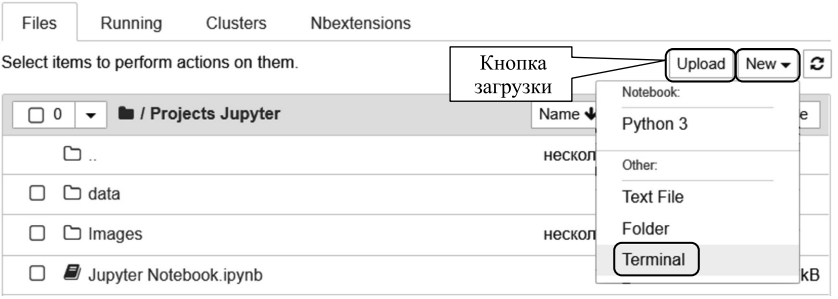


Рис. 9.39. Вкладка *Files*

Ранее уже рассматривались вопросы создания файлов и папок в *Jupyter Notebook* с помощью кнопки *New* и последующим выбором соответствующей команды (рис. 9.39). Здесь находится также команда *Terminal* (*Терминал*), которая запускает терминал вашей операционной системы в браузере, где можно выполнять необходимые для работы команды.

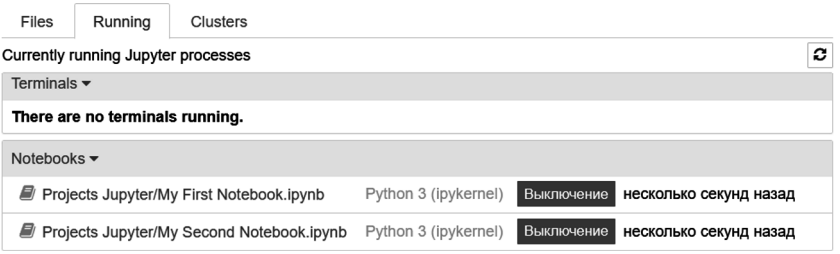


Рис. 9.40. Вкладка *Running*

Вкладка *Running* (рис. 9.40) отображает работающие в текущий момент ноутбуки и используемые ими ядра. Эта вкладка окажется довольно полезной, когда требуется выключить сервер, но нужно убедиться, что все данные сохранены.

Контрольные вопросы

1. Что такое *Jupyter Notebook*?
2. Как называются файлы, созданные в Jupyter Notebook, и какое они имеют расширение?
3. Что такое Markdown?

4. Что такое LaTeX?

5. Что понимают под ядром (Kernel) в *Jupyter Notebook*?

6. Какие типы ячеек можно использовать в *Jupyter Notebook*?

7. Каким образом можно запустить *Jupyter Notebook*?

8. Из каких элементов состоит окно браузера после запуска *Jupyter*?

9. Как в *Jupyter Notebook* создать папку?

10. Какие типы файлов можно создать в *Jupyter Notebook*?

11. Как в *Jupyter Notebook* создать новый ноутбук?

12. Каким образом в *Jupyter Notebook* переименовать папку/файл?

13. Какие режимы работы с ячейками существуют и как узнать текущий режим работы ячейки?

14. Каким образом можно запустить ячейку на выполнение?

15. Что означает нелинейная последовательность выполнения команд? Приведите пример.

16. Как добавить новую ячейку с помощью панели инструментов и комбинацией клавиш?

17. Как поменять порядок следования ячеек в ноутбуке?

18. Что такое контрольная точка?

19. Перечислите основные «горячие» клавиши, используемые при работе в *Jupyter Notebook* и их назначение.

20. Как посмотреть список «горячих» клавиш *Jupyter Notebook*?

21. Как получить подсказку по командам и используемым объектам в программе?

22. Перечислите кнопки панели инструментов *Jupyter Notebook* и их назначение.

23. Как пронумеровать строки в ячейке?

24. Как изменить тип ячейки?

25. Охарактеризуйте команды меню Kernel.

26. Приведите примеры разметки Markdown, используемой для форматирования текста и заголовков.

27. Как в разметке Markdown создать таблицу?

28. Как добавить в ноутбук ссылку на Интернет-ресурс?

29. Как добавить изображение в ноутбук?

30. Как добавить формулу в ноутбук?

31. Какие режимы вставки формул вам известны? Приведите примеры их использования.

32. Как установить в *Jupyter Notebook* внешнюю библиотеку?

33. Что такое магическая команда и какие виды магических команд вы знаете?

34. Приведите примеры использования магических команд.

35. Что такое виджеты и какие виджеты часто используются?

36. Приведите примеры использования виджетов в программе.

37. Что понимают под расширением для *Jupyter Notebook*?

38. Какие расширения позволяют повысить продуктивность использования *Jupyter Notebook*?

39. Как изменить тему оформления в *Jupyter Notebook*?

40. Как можно импортировать файлы в *Jupyter Notebook*?

Задания

Во всех вариантах заданий подготовьте ноутбук с отчетом к лабораторной работе, в который включите:

1) текст задания своего варианта (выражение для нахождения значения функции подготовьте в формате LaTeX);

2) рисунок с блок-схемой решения задачи, подготовленный с использованием графического редактора, содержащего инструменты для создания блок-схем, например *Microsoft Visio* или бесплатный онлайн-редактор *Draw.io*, доступный по адресу

https://www.draw.io/

3) программу решения задачи, подготовленную в двух вариантах: в первом для ввода данных используйте функцию input(), во втором для задания значений входных переменных используйте виджеты;

4) расчет для трех тестовых наборов входных данных, выполненный с использованием компьютерных программ для математических и технических расчетов, например *MathCAD*, *WolframAlpha* или *SMath Studio* (скрин результатов расчета из программы);

5) расчет контрольных значений для пяти наборов входных данных, включающий наборы из п. 4, представленный в виде таблицы, округлив результат вычисления значения функции до четырех знаков после запятой.

При подготовке ноутбука используйте форматирование текста и заголовки.

Переконвертируйте созданный ноутбук в *pdf*-файл.

На рис. 9.41 представлена блок-схема для нахождения значения функции y, заданной выражением

$$y = \begin{cases} \sum\limits_{k=1}^{10} x^k \cos\dfrac{k\pi}{3}, & \text{если } x \geqslant a; \\ \prod\limits_{m=1}^{6} (a^m + x^m), & \text{если } x < a. \end{cases}$$

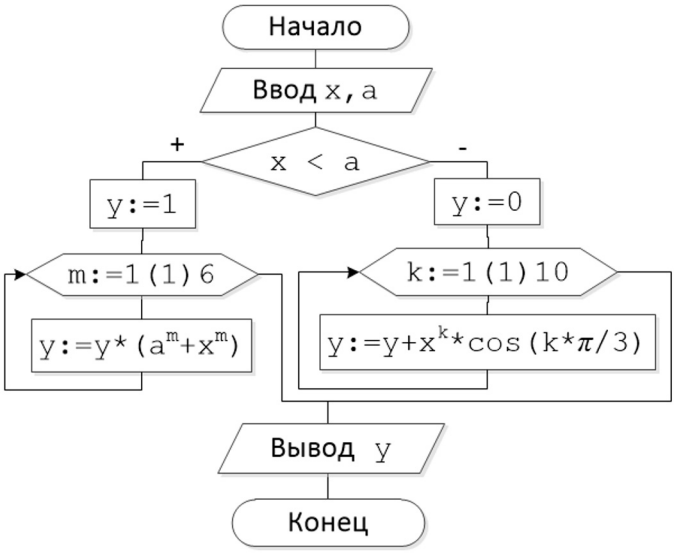


Рис. 9.41. Блок-схема решения задачи

$$y(x,a) := \text{if}\left[x < a,\ \prod_{m=1}^{6}\left(a^m + x^m\right),\ \sum_{k=1}^{10}\left(x^k \cdot \cos\left(\frac{k\cdot\pi}{3}\right)\right)\right]$$

$y(-2,1.5) = -560131.6026$ $\quad$ $y(-1,-1) = -0.5$

$y(2.6,-1.3) = -12804.5854$ $\quad$ $y(0,1.2) = 46.0051$

Рис. 9.42. Расчет тестовых значений в *MathCAD*

На рис. 9.42 представлены результаты расчета значений, выполненные в математическом пакете *MathCAD*, а на рис. 9.43 — результаты работы программы из *Jupyter Notebook*, представленные в виде таблицы.

x	a	y(x, a)
-2	1.5	-560131.6026
2.6	-1.3	-12804.5854
-1	-1	-0.5
0	1.2	46.0051
1.2	-2.3	-7.2686

Рис. 9.43. Расчет контрольных значений

Вариант 1. Вычислить

$$y = \begin{cases} \sum\limits_{k=1}^{10} x^k \sin\dfrac{k\pi}{4}, & \text{если } x \geqslant a; \\ \prod\limits_{m=1}^{5} (a^m - x^m), & \text{если } x < a. \end{cases}$$

Вариант 2. Вычислить произведение

$$P = \prod_{i=1}^{10} \frac{i(i+a)}{i^2+a^2} + \prod_{k=2}^{4} \frac{k^3}{k+a}.$$

Вариант 3. Вычислить сумму

$$S = \sum_{n=0}^{100} (-1)^n \frac{x^{2n}}{(2n)!}.$$

Вариант 4. Вычислить

$$y = \frac{\sin x + 2}{3 + \cos x} \sum_{n=0}^{20} a x^n; \quad a = \begin{cases} 2n, & \text{если } x \leqslant 0{,}5; \\ n/2, & \text{если } x > 0{,}5. \end{cases}$$

Вариант 5. Вычислить

$$z = \sum_{k=0}^{10} \ln x \sin k(x-a); \quad a = \begin{cases} \pi/4, & \text{если } x \leqslant 1; \\ \pi, & \text{если } x > 1. \end{cases}$$

Вариант 6. Вычислить

$$y = \begin{cases} \sum_{k=1}^{10} x^k a^k, & \text{если } x \geqslant a; \\ \prod_{m=1}^{8} (a^m - x^m), & \text{если } x < a. \end{cases}$$

Вариант 7. Вычислить сумму

$$\alpha = \sum_{k=1}^{10} (k + a \ln x); \quad a = \begin{cases} -1, & \text{если } x \leqslant 1; \\ 1, & \text{если } x > 1. \end{cases}$$

Вариант 8. Вычислить

$$z = \begin{cases} \prod\limits_{n=1}^{8} \left(\dfrac{x}{n}\right)^n, & \text{если } x \leqslant 2; \\ \sum\limits_{n=0}^{5} (1 + xn), & \text{если } x > 24. \end{cases}$$

Вариант 9. Вычислить произведение

$$y = \prod_{n=1}^{8} \left(p - \frac{x^n}{2n+1} \right); \quad p = \begin{cases} 1, & \text{если } n \leqslant 5; \\ 2, & \text{если } n > 5. \end{cases}$$

Вариант 10. Вычислить

$$y = \frac{ax^2}{\sqrt{x+a}}; \quad a = \begin{cases} \sum\limits_{k=1}^{12} \dfrac{2kx}{x+k^2}, & \text{если } x < 1; \\ 1, & \text{если } x \geqslant 1. \end{cases}$$

Вариант 11. Вычислить сумму

$$S = \sum_{n=0}^{20} \frac{2\,(x-1)^{2n-1}}{(2n+1)\,(x+1)^{2n+1}}.$$

Вариант 12. Вычислить произведение

$$\lambda = \prod_{n=1}^{5} \left(\frac{1}{n} + a \sin x \right); \quad a = \begin{cases} 1, & \text{если } x \geqslant 0; \\ -1, & \text{если } x < 0. \end{cases}$$

Вариант 13. Вычислить

$$z = \begin{cases} \dfrac{x+1}{2} \displaystyle\sum_{n=1}^{10} \left(\frac{x}{n} \right)^n, & \text{если } x \leqslant 2; \\ \dfrac{\sin x + \cos x}{2 + \sin x} \displaystyle\prod_{n=0}^{15} \left(1 + \frac{x}{n+2} \right), & \text{если } x > 2. \end{cases}$$

Вариант 14. Вычислить произведение

$$P = \prod_{n=1}^{5} \left(n + \frac{\sin x}{n} \right).$$

Вариант 15. Вычислить сумму

$$S = \sum_{n=0}^{20} \frac{x^{2n+1}}{(2n+1)!}.$$

Вариант 16. Вычислить

$$z = \begin{cases} \displaystyle\prod_{n=1}^{8} \left(\frac{x}{n} \right)^n, & \text{если } x \leqslant 1; \\ \displaystyle\sum_{n=1}^{5} (3 + xn), & \text{если } x > 1. \end{cases}$$

Вариант 17. Вычислить

$$y = \frac{x+2}{1+\cos x}\sum_{n=1}^{10} a\,(1+x^n); \quad a = \begin{cases} 2n, & \text{если } x \leqslant 1; \\ n/2, & \text{если } x > 1. \end{cases}$$

Вариант 18. Вычислить сумму

$$S = \sum_{k=0}^{20} \frac{2+\sin\left(\frac{\pi}{k}x\right)}{3+\cos\left(\frac{\pi x}{k+2}\right)}.$$

Вариант 19. Вычислить

$$z = \frac{a+1}{a+2{,}5\cdot 10^{-3}} \sum_{n=0}^{15} \frac{a^{2n}}{(n+1)!}.$$

Вариант 20. Вычислить

$$z = \begin{cases} \prod\limits_{n=1}^{5} \left(n + \dfrac{\sin x}{n}\right), & \text{если } x \leqslant 2; \\ \sum\limits_{n=0}^{5} (1+xn), & \text{если } x > 2. \end{cases}$$

Вариант 21. Вычислить произведение

$$\lambda = \prod_{n=1}^{10} \left(1 + \frac{ax}{n+2}\right); \quad a = \begin{cases} 1, & \text{если } x \geqslant 0; \\ -1, & \text{если } x < 0. \end{cases}$$

Вариант 22. Вычислить сумму

$$z = \sum_{n=0}^{10} \frac{(-1)^{2n} x^{2n}}{(2n+1)!}.$$

Вариант 23. Вычислить

$$z = \sum_{k=1}^{15} x\cos k(x+a); \quad a = \begin{cases} \pi/3, & \text{если } x \leqslant 1; \\ \pi, & \text{если } x > 1. \end{cases}$$

Вариант 24. Вычислить

$$y = \frac{ax^2}{\sqrt{|x+a|}} \sum_{k=1}^{10} \frac{2kx}{x+k}; \quad a = \begin{cases} 1, & \text{если } x < 1; \\ -1, & \text{если } x \geqslant 1. \end{cases}$$

Вариант 25. Вычислить произведение

$$P = \prod_{n=0}^{10} \left(2n + \frac{\sin x}{n+1} \right).$$

Краткий словарь терминов Python

Автокомплит (autocomplite) — автозаполнение. Обычно автозаполнение выполняется клавишей Tab и автоматически завершает вводимое слово.

Брекпойнт (breakpoint) — точка останова — преднамеренное прерывание выполнения программы, при котором выполняется вызов отладчика.

Геттер (от англ. getter — получатель) — специальный метод, позволяющий получить данные, доступ к которым напрямую ограничен.

Инлайнинг (inlining) — встраивание объекта (модуля и пр.).

Интроспекция (introspection) — возможность запросить тип и структуру объекта во время выполнения программы.

Лаг (жаргонное, чаще лог, log) — запись в журнале, обычно для отладки программы.

Репозитории (repository) — классы или компоненты, которые содержат логику, необходимую для доступа к источникам данных.

Рефакторинг (refactoring) — метод изменения приложения (кода или архитектуры), чтобы оно работало одинаково снаружи, но внутренне улучшилось.

Сеттер (от англ. setter — установщик) — метод, который используется для установки значения свойства.

Скоуп (область видимости, scope) — текстовая область программы Python, в которой пространство имен доступно напрямую.

Фреймворк (неологизм от framework — остов, каркас, рама, структура) — программная платформа, определяющая структуру программной системы.

Фолдинг (folding) — функция, которая позволяет скрывать блоки кода.

Литература

1. Лутц Марк. Изучаем Python. Т. 1, 5-е изд. Пер. с англ. — СПб.: ООО «Диалектика», 2019. — 832 с.

2. Постолит А.В. Python. Django и PyCharm для начинающих. — СПб.: БХВ-Петербург, 2021. — 464 с.

3. Федоров Д.Ю. Программирование на языке высокого уровня Python: учеб. пособие для прикладного бакалавриата. 2-е изд., перераб. и доп. — М.: Юрайт, 2019. — 161 с.

4. https://coderlessons.com/tutorials/python-technologies/jupyter/uchebnik-po-jupyter

5. https://younglinux.info/python/pycharm

6. https://www.jetbrains.com/ru-ru/pycharm/learn/

Оглавление

Предисловие ... 3
Введение ... 5
1. Установка и запуск среды PyCharm ... 7
Установка среды PyCharm ... 7
Начало работы ... 8
Выполнение программы ... 10
Дополнительные варианты выполнения команд ... 11
Переименование переменных ... 11
Базовая настройка IDE PyCharm ... 12
Пример программы ... 12
2. Инструкция по началу работы в среде PyCharm ... 14
На каких языках поддерживается PyCharm? ... 14
На каких платформах я могу запустить PyCharm? ... 14
Шаг 1. Откройте/создайте проект в PyCharm ... 14
Шаг 2. Ознакомьтесь с интерфейсом ... 15
Шаг 3. Настройте свою среду ... 16
Шаг 4. Сделайте свой код аккуратным ... 17
Шаг 5. Создание кода ... 18
3. Перевод и назначение инструментов среды PyCharm ... 19
Information ... 19
Окно среды ... 19
Панель проверки ... 21
Панель Problems ... 21
Панель Terminal ... 22
Панель Python Packages ... 23
Панель Python Console ... 23
Верхняя правая часть окна среды ... 24
Верхняя панель ... 25
Меню File ... 25

Меню Edit 27
Меню View 29
Меню Navigate 30
Меню Code 32
Меню Refactor 34
Меню Run 36
Меню Tools 38
Меню VCS 39
Меню Window 40
Меню Help 41
4. Горячие клавиши PyCharm 44
5. Основные элементы и структура программы на языке Python 50
Комментарии 51
Литералы 51
Знаки пунктуации. Базовые операторы 53
Арифметические операторы 54
Операторы сравнения 54
Операторы присваивания 55
Побитовые операторы 55
Логические операторы 57
Операторы принадлежности 57
Операторы тождественности 57
Приоритет выполнения операторов 57
Ленивые логические выражения 58
Идентификаторы 59
Синтаксическая структура программы 60
Инструкции ввода и вывода данных 61
Модуль matc 63
6. Отладка 66
Понятие об отладке программ 66
Причины и типы ошибок 67
Способы и средства отладки 68
Отладка программ в среде PyCharm 70
7. Руководство PEP 8 по написанию кода 75
Внешний вид кода 75
Пробелы в выражениях и инструкциях 78

Негласная договоренность об именах ... 81
Комментарии ... 81
Философия программирования ... 82
8. Наиболее часто реализуемые действия в среде PyCharm ... 84
Назначение меню Help/Find Action ... 84
Установка Python ... 84
Импорт/добавление существующего файл Python в проект PyCharm ... 88
Quick фиксы (действия по быстрому исправлению кода) в PyCharm ... 88
Навигация с помощью курсора ... 88
Настройка собственного стиля кода ... 89
Настройка PyCcarm + GitHub ... 89
Просмотр структуры модуля ... 90
9. Программирование на Python в *Jupyter Notebook* ... 91
Общие сведения о *Jupyter Notebook* ... 91
Установка *Jupyter Notebook* ... 92
Терминология *Jupyter Notebook* ... 95
Запуск Jupyter ... 97
Создание и переименование папки ... 100
Создание и сохранение блокнота ... 100
Интерфейс *Jupyter Notebook* ... 106
Горячие клавиши Jupyter Notebook ... 106
Получение справочной информации ... 107
Панель инструментов ... 108
Строка меню Jupyter Notebook ... 109
Приемы документирования в *Jupyter Notebook* ... 111
Приемы форматирования текста ... 111
Добавление таблиц ... 114
Вставка ссылок на Интернет-ресурсы ... 114
Вставка медиаконтента ... 115
Работа с формулами ... 115
Дополнительные возможности кодовых ячеек ... 119
Установка сторонних пакетов в Jupyter Notebook ... 119
Комментирование кода ... 120
Вывод значений ... 121
Магические команды ... 121

Интерактивное управление с помощью виджетов 123
Дополнительные настройки *Jupyter Notebook* 126
Расширения 126
Изменение темы оформления 128
Управление папками и ноутбуками в Jupyter 129
Контрольные вопросы 130
Задания 132
Краткий словарь терминов Python 138
Литература 139