

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
**Федеральное государственное автономное образовательное
учреждение высшего образования
«Севастопольский государственный университет»**

Институт радиоэлектроники и интеллектуальных технических систем
Кафедра «Информатика и управление в технических системах»



МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к выполнению лабораторной работы
**«Программные средства обработки и анализа данных
на Python. Основы Python»**

по дисциплине
«Обработка данных в автоматизированных системах»

*для студентов очной формы обучения направления 27.03.04
«Управление в технических системах» (профиль подготовки
«Интеллектуальные робототехнические системы»)*

Версия 12082024

Севастополь — 2024

УДК 004.6

Методические указания к выполнению лабораторной работы «Программные средства обработки и анализа данных на Python. Основы Python» по дисциплине «Обработка данных в автоматизированных системах» для студентов очной формы обучения направления 27.03.04 «Управление в технических системах» (профиль подготовки «Интеллектуальные робототехнические системы») / Сост. Альчаков В.В. — Севастополь: Изд-во ФГАОУ ВО «Севастопольский государственный университет», 2024. — с. 56.

Методические указания:

рассмотрены и рекомендованы к изданию решением кафедры «Информатика и управление в технических системах», протокол № 1 от 30.08.2024 г.;

допущены учебно-методическим центром ФГАОУ ВО «Севастопольский государственный университет» в качестве методических указаний.

Рецензент:

Крамарь В.А., д-р техн. наук., профессор, профессор кафедры «Информатика и управление в технических системах» СевГУ.

© СевГУ, 2024

© Альчаков В.В., 2024

Содержание

Цель работы.....	4
Порядок выполнения и задание на работу.....	4
Требования к отчёту по лабораторной работе.....	4
Основные теоретические сведения.....	6
Установка пакета Anaconda.....	7
Установка Python, PyCharm.....	16
Работа с блокнотом Jupyter Notebook.....	23
Создание проекта в PyCharm.....	28
Установка пакетов с помощью менеджера пакетов pip.....	32
Основы Python.....	35
Библиотека math.....	47
Работа с файлами.....	49
Варианты задания.....	50
Пример выполнения работы.....	51
Контрольные вопросы.....	54
Список использованных источников.....	54
Полезные ссылки.....	55

Цель работы

Изучить современные программные средства обработки и анализа данных на Python. Научиться выполнять установку и настройку рабочего окружения. Освоить работу с блокнотами Jupyter Notebook и с IDE PyCharm. Научиться устанавливать дополнительные библиотеки. Изучить основы синтаксиса языка Python, освоить работу с файлами в Python, вычисление математических выражений, изучить библиотеку math.

Порядок выполнения и задание на работу

1. Ознакомиться с основными современными программными средствами для анализа данных.
2. Выполнить установку пакета Anaconda.
3. Изучить структуру блокнота Jupyter Notebook.
4. Изучить горячие клавиши для работы с блокнотом Jupyter Notebook.
5. Научиться сохранять и загружать блокноты Jupyter Notebook с диска.
6. Изучить возможности пакета Anaconda для администрирования пакетов.
7. Выполнить установку Python и IDE PyCharm.
8. Изучить структуру проекта PyCharm.
9. Научиться сохранять и загружать проекты PyCharm с диска.
10. Изучить возможности PyCharm для администрирования пакетов.
11. Изучить утилиту pip для администрирования пакетов.
12. Используя Jupyter Notebook и IDE PyCharm записать скрипт для расчета значения математической функции в заданном диапазоне изменения аргумента. Входные параметры должны читаться из файла, результат вычисления функции — выводиться в файл. Алгоритм расчета математической функции должен быть реализован с помощью инструкции def.

Требования к отчёту по лабораторной работе

Отчёт о выполненной лабораторной работе должен содержать:

- титульный лист;
- цель лабораторной работы;
- основные положения;
- ход работы;
- выводы по работе;
- приложения (листинги исходных кодов программ).

Отчёт составляется каждым обучающимся индивидуально и должен соответствовать варианту задания, назначенного преподавателем.

В отчёте к данной лабораторной работе необходимо привести описание структуры блокнота Jupyter Notebook и проекта PyCharm, горячие клавиши для работы с блокнотом Jupyter Notebook, сравнительную характеристику двух инструментальных средств для анализа данных (Jupyter Notebook vs PyCharm). Также в отчёте должны быть представлены форматы команд, используемые для установки дополнительных библиотек. В выводе по работе сделать выбор инструментального средства и обосновать своё решение.

Для выбранной платформы разработки решить задачу табулирования функции на заданном интервале с помощью скрипта на языке Python. Подробно описать фрагменты кода и обосновать выбор тех или иных функций и библиотек. При выводе данных в текстовый файл использовать форматированный вывод числовых значений параметров, ограничить точность вывод двумя знаками после запятой для параметра x и четыре знака для значения функции $f(x)$.

Основные теоретические сведения

В современном мире данные и информация (интерпретированные данные) играют важную роль при решении ряда задач, в том числе задач управления, прогнозирования, выработки стратегических решений. Объемы доступных данных растут и обрабатывать их вручную становится просто невозможно. На помощь приходят различные инструменты обработки и анализа данных, владение которыми дает значительные конкурентные преимущества. Среди самых популярных инструментов можно выделить:

- Microsoft Excel / LibreOffice Calc – для работы с табличными данными;
- Tableau – для интерактивной визуализации и бизнес-аналитики;
- Statistica – для статического и графического анализа, прогнозирования и обработки данных;
- RapidMiner – платформа для анализа данных;
- Orange – набор инструментов для визуализации данных, машинного обучения и интеллектуального анализа данных;
- Power BI – комплексный инструмент бизнес-анализа от компании Microsoft.

Кроме готовых инструментов, существует возможность создавать свои собственные с помощью языков программирования C++ / C# / Java / Scala и т. д. Также имеются специализированные языки R и Python, специально разработанные для удобства работы с данными. В рамках курса основное внимание будет уделено языку Python. Поскольку именно этот язык получил более широкое (по сравнению с R) распространение, относительно прост в написании кода, располагает сотнями прикладных библиотек, предназначенных для обработки и анализа данных.

За последние десятилетия (язык Python появился в 1991 году) Python превратился из специализированного языка научных расчетов в один из самых важных языков, применяемых в науке о данных, в машинном обучении и разработке ПО общего назначения в академических учреждениях и промышленности.

Установка пакета Anaconda

Инструкция по установке для ОС Linux

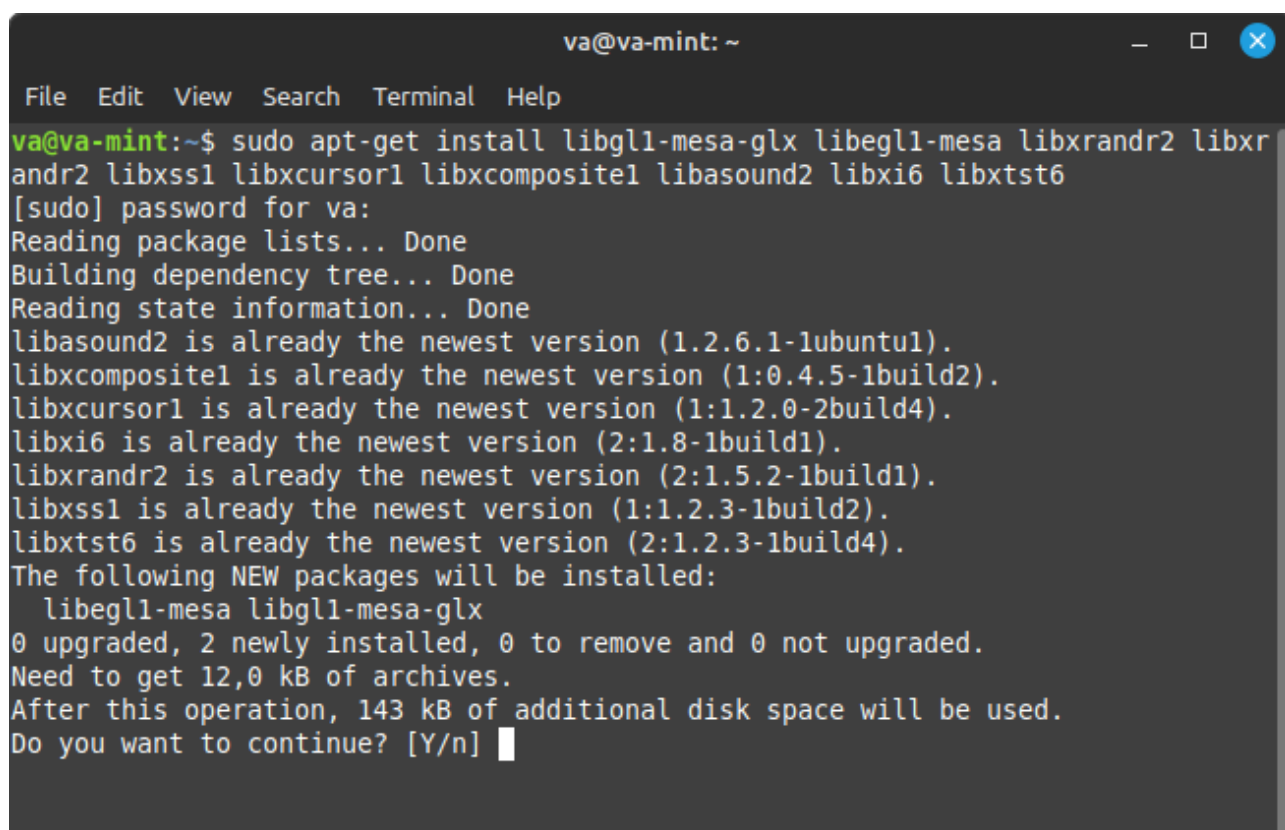
Актуальная инструкция установки пакета Anaconda для операционной системы семейства Linux доступна по ссылке <https://docs.anaconda.com/anaconda/install/linux/>

Далее процесс установки будет описан на примере ОС Linux Mint 21.1 Cinnamon Edition.

Шаг 1. Открыть окно терминала и выполнить установку зависимостей с помощью команды

```
sudo apt-get install libgl1-mesa-glx libegl1-mesa libxrandr2 libxrandr2 libxss1 libxcursor1 libxcomposite1 libasound2 libxi6 libxtst6
```

В ходе установки подтвердить запрос программы установщика, введя символ **Y** в окне терминала.



```
va@va-mint: ~  
File Edit View Search Terminal Help  
va@va-mint:~$ sudo apt-get install libgl1-mesa-glx libegl1-mesa libxrandr2 libx  
andr2 libxss1 libxcursor1 libxcomposite1 libasound2 libxi6 libxtst6  
[sudo] password for va:  
Reading package lists... Done  
Building dependency tree... Done  
Reading state information... Done  
libasound2 is already the newest version (1.2.6.1-lubuntu1).  
libxcomposite1 is already the newest version (1:0.4.5-1build2).  
libxcursor1 is already the newest version (1:1.2.0-2build4).  
libxi6 is already the newest version (2:1.8-1build1).  
libxrandr2 is already the newest version (2:1.5.2-1build1).  
libxss1 is already the newest version (1:1.2.3-1build2).  
libxtst6 is already the newest version (2:1.2.3-1build4).  
The following NEW packages will be installed:  
  libegl1-mesa libgl1-mesa-glx  
0 upgraded, 2 newly installed, 0 to remove and 0 not upgraded.  
Need to get 12,0 kB of archives.  
After this operation, 143 kB of additional disk space will be used.  
Do you want to continue? [Y/n]
```

Рис. 1 — Установка зависимостей

Шаг 2. Дождаться окончания процесса установки зависимостей, после чего перейти на страницу загрузки пакета Anaconda для Linux по ссылке <https://www.anaconda.com/download/#linux>

На странице загрузке нажать кнопку **Download** (Рис. 2)

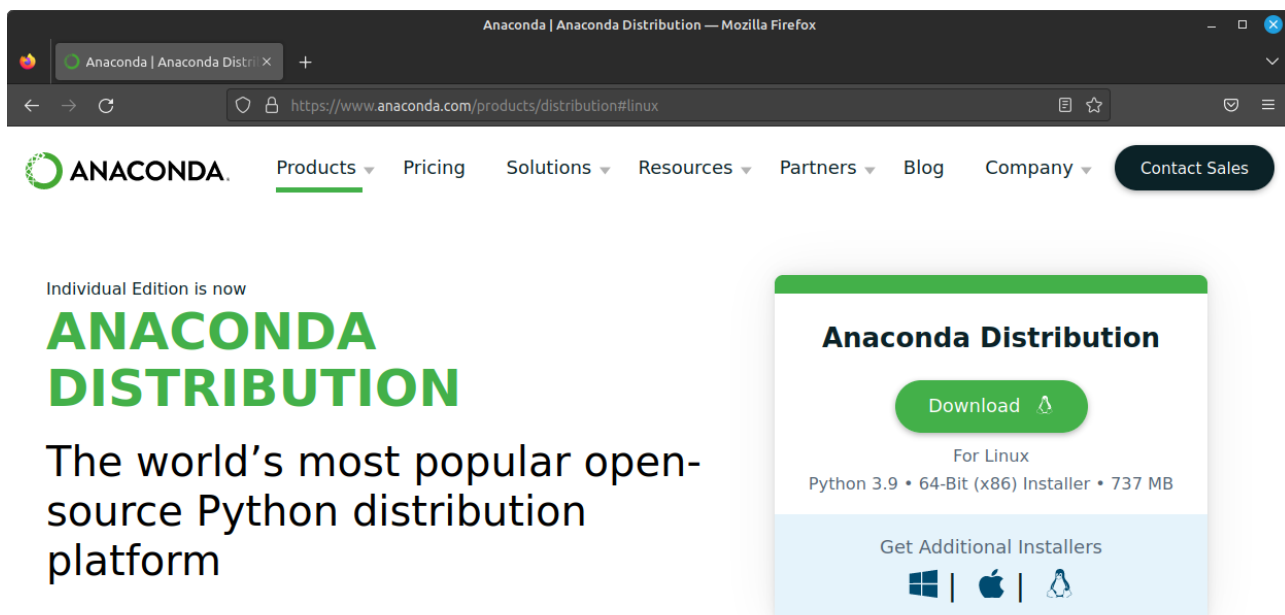


Рис. 2 — Страница загрузки пакета Anaconda

Шаг 3. По завершению загрузки, перейти в папку Downloads и открыть содержимое папки в окне терминала. Для этого вызвать контекстное меню и выбрать опцию **Open in Terminal** (Рис. 3). После чего выполнить команду `bash Anaconda3-2022.10-Linux-x86_64.sh`

Внимание: имя файла установщика может отличаться и должно соответствовать актуальной версии пакета. Содержимое папки может быть получено с помощью команды `ls` (Рис. 4).

После запуска команды необходимо нажать кнопку **Enter** и следовать дальнейшим указаниям установщика (ознакомиться с содержанием лицензионного соглашения). Дождаться окончания установки пакета.

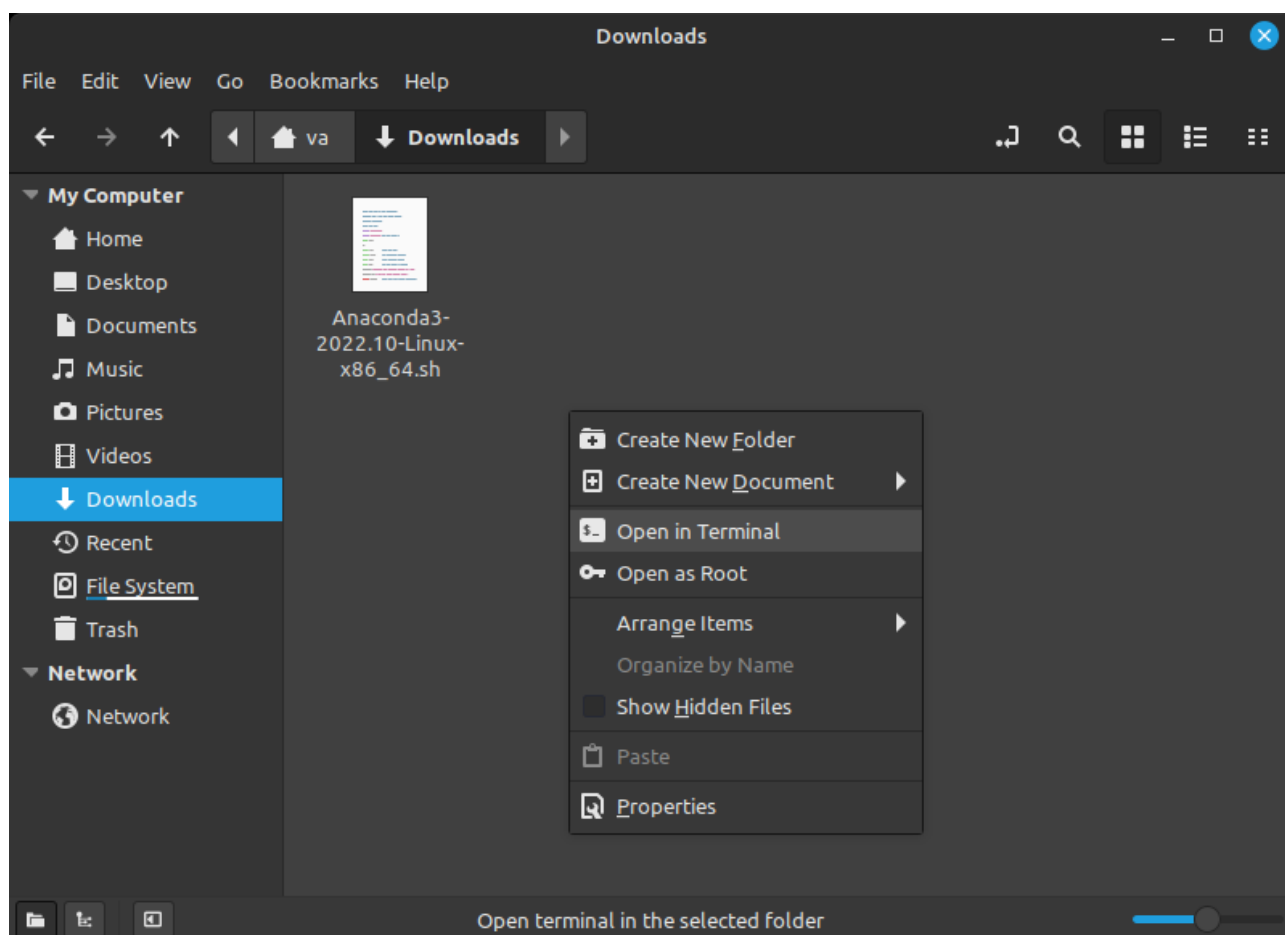


Рис. 3 — Открытие папки в окне терминала

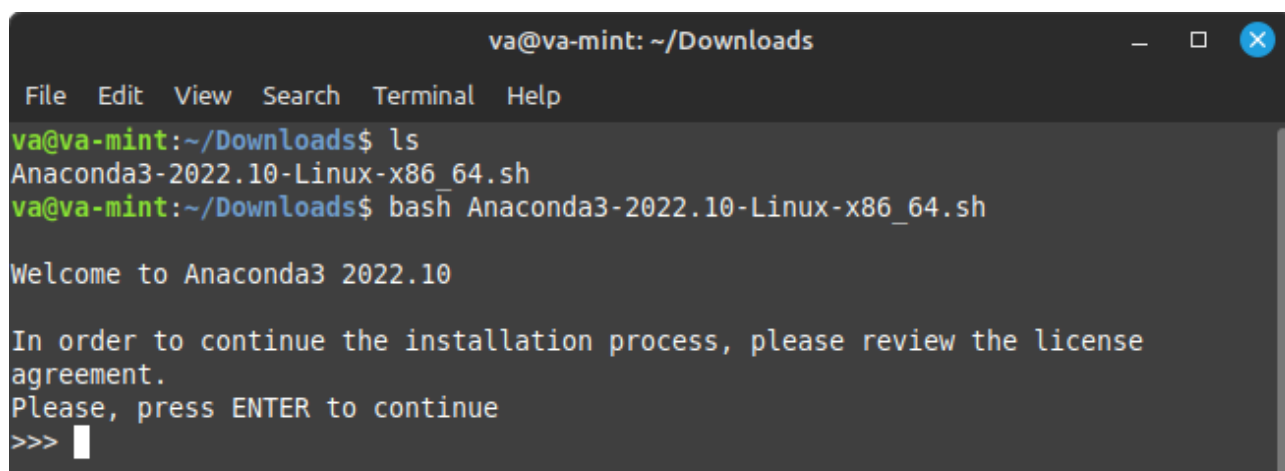


Рис. 4 — Запуск скрипта установки пакета Anaconda

Шаг 4. По окончании установки может быть запущена оболочка, которая используется для удобства запуска отдельных программ пакета — Anaconda Navigator (далее навигатор) (Рис. 5). Для запуска навигатора в окне терминала необходимо выполнить команду `anaconda-navigator`

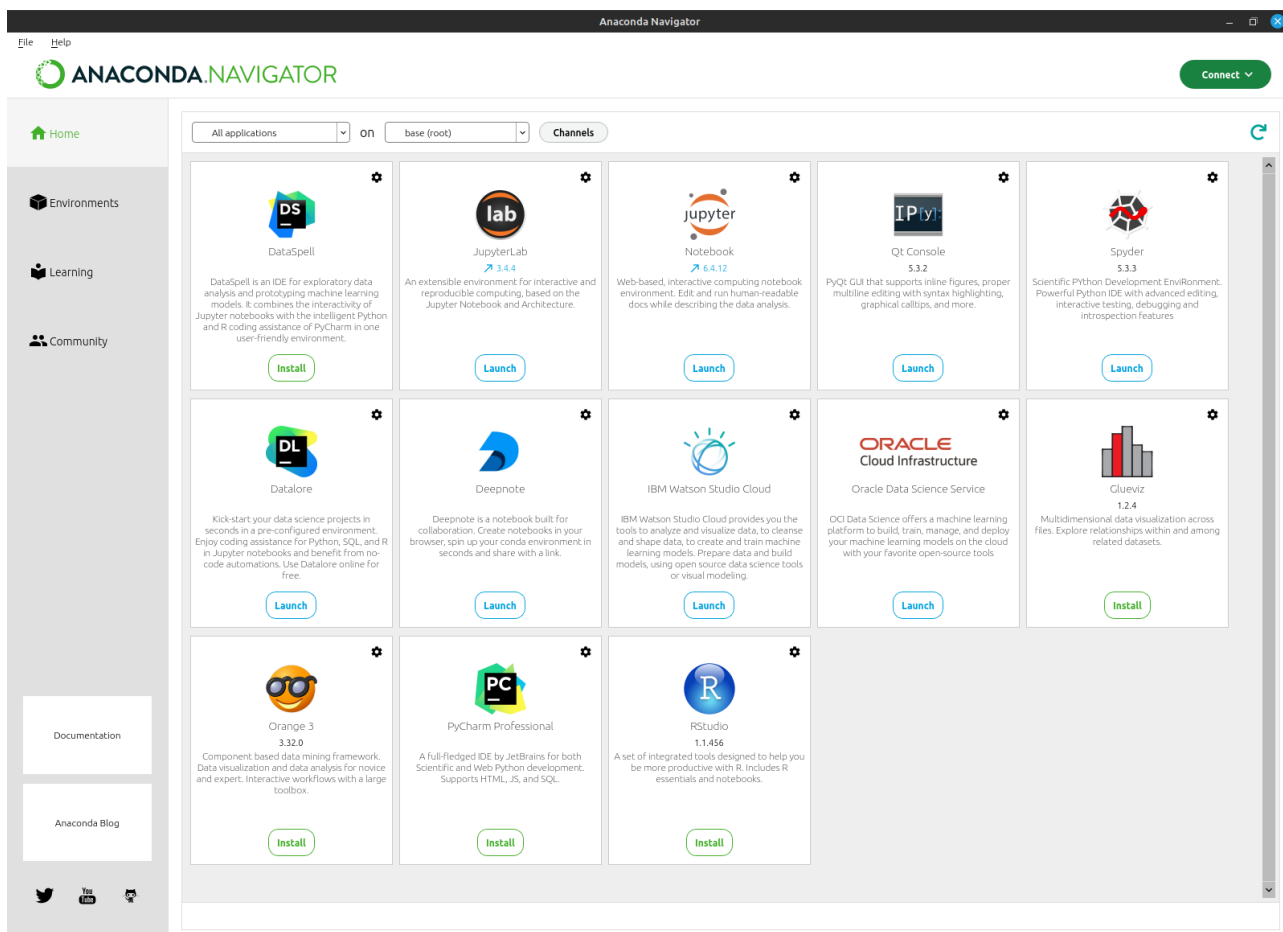


Рис. 5 — Окно Anaconda Navigator

Как сказано выше, навигатор может быть использован для быстрого доступа к инструментам, входящим в состав пакета Anaconda, в том числе и к Jupyter Notebook. Для запуска выбранного инструмента необходимо нажать кнопку **Launch** в соответствующей панели (Рис. 6)

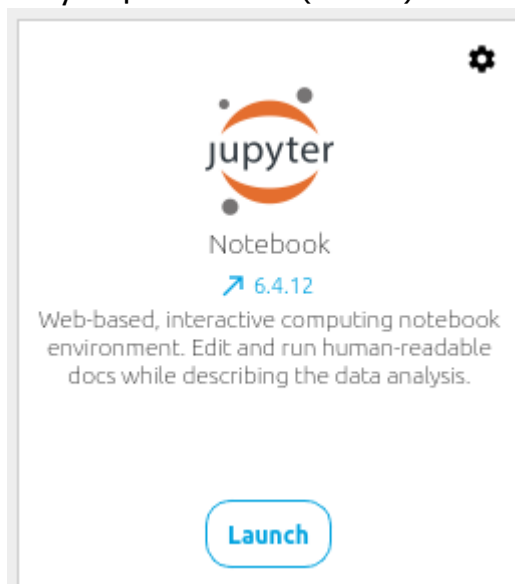


Рис. 6 — Панель запуска Jupyter Notebook

Инструкция по установке для ОС Windows

Актуальная версия установочного пакета Anaconda для операционной системы семейства Windows доступна по ссылке <https://www.anaconda.com/products/distribution>

Шаг 1. Скачать установочный пакет, нажав на кнопку Download на странице загрузки

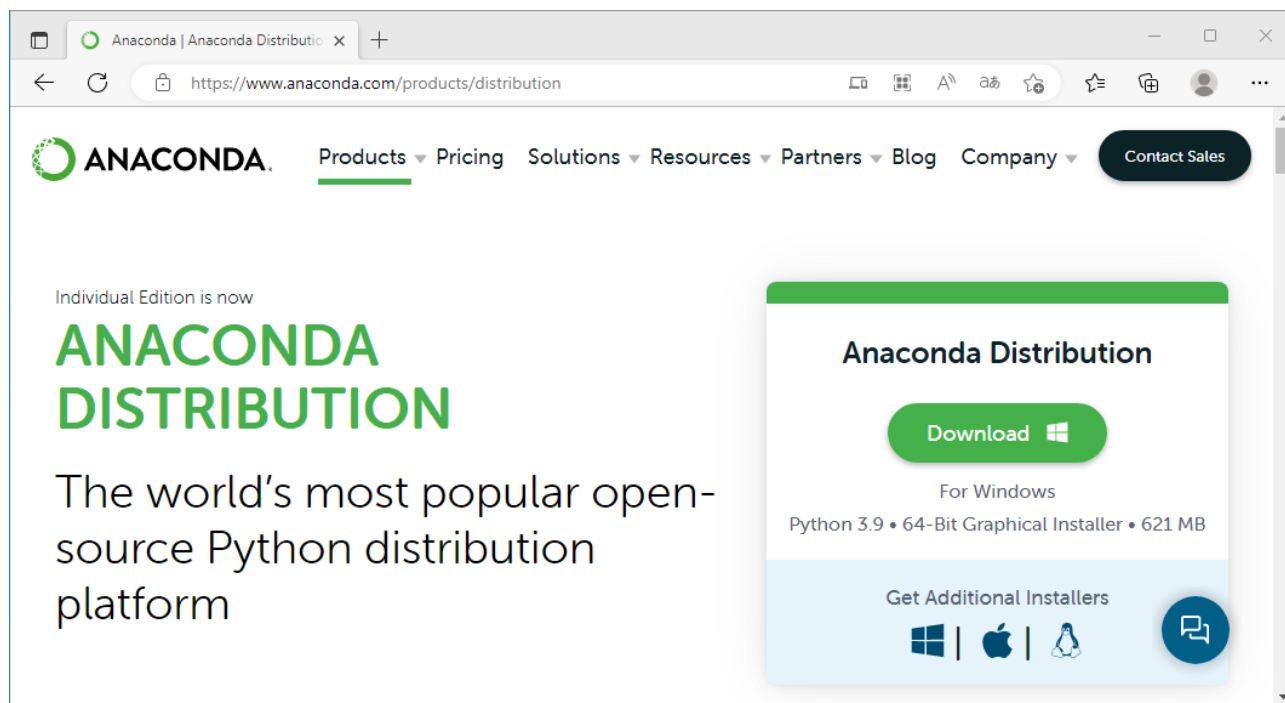


Рис. 7 — Страница загрузки пакета Anaconda для ОС Windows

Шаг 2. По окончании загрузки запустить файл **Anaconda3-2022.10-Windows-x86_64.exe** на исполнение.

Внимание: в зависимости от версии пакета имя скачанного файла может отличаться.

Далее необходимо следовать указаниям программы-установщика. Параметры установки рекомендуется оставить без изменения.

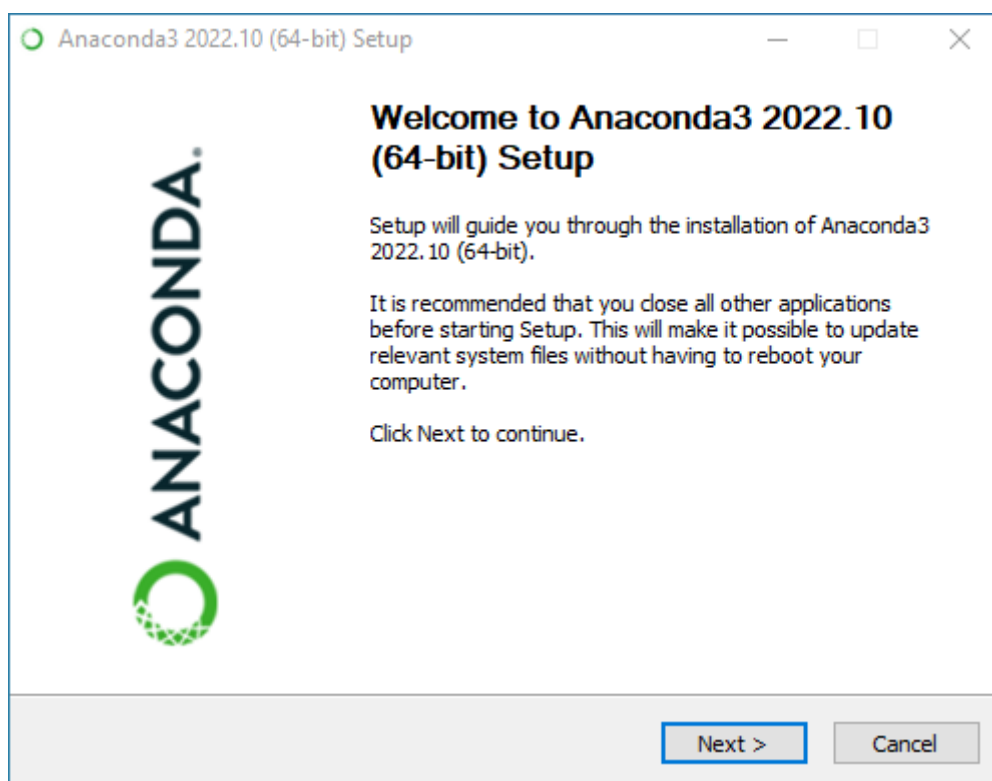


Рис. 8 — Страница приветствия программы-установщика Anaconda

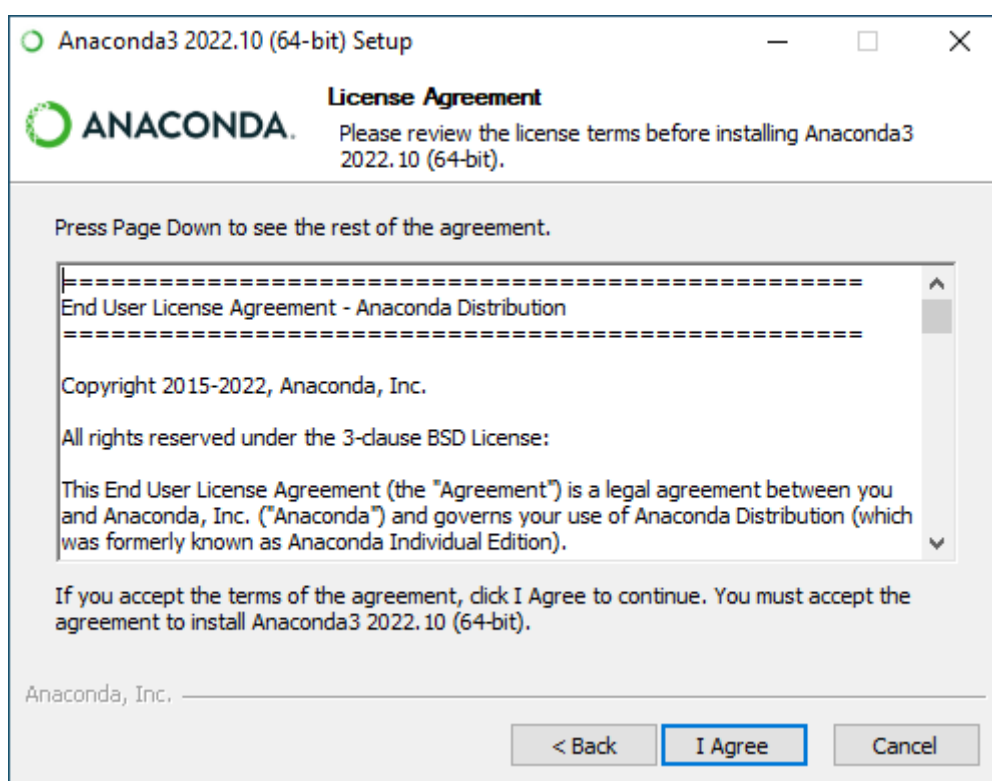


Рис. 9 — Страница лицензионного соглашения Anaconda

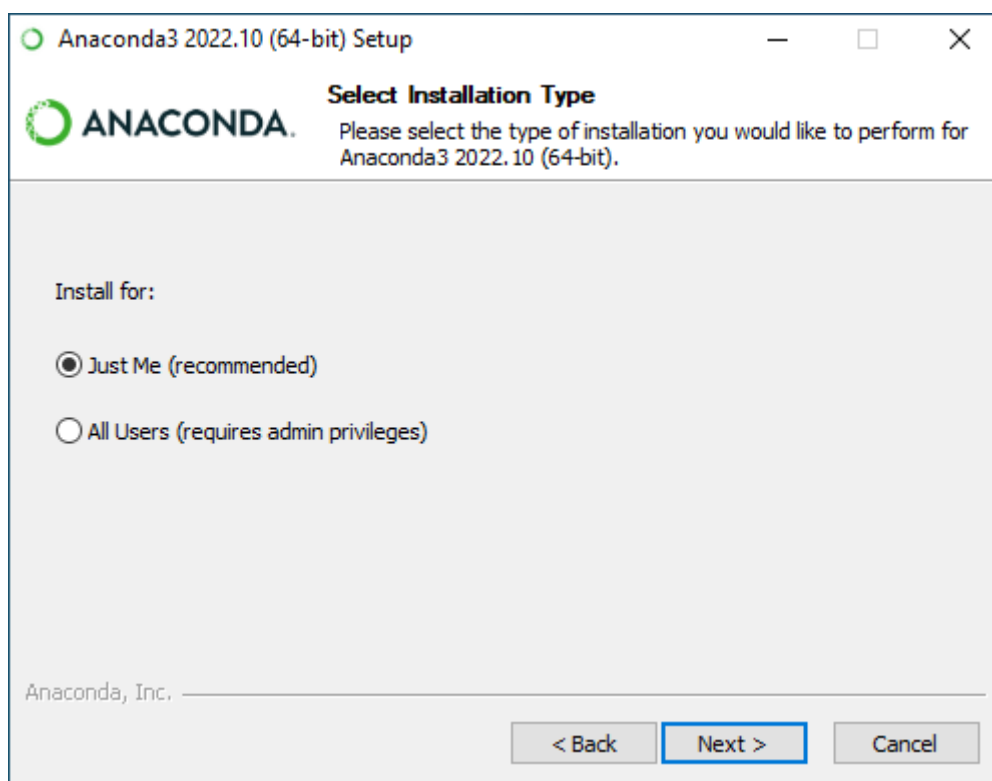


Рис. 10 — Страница конфигурации параметров установки Anaconda

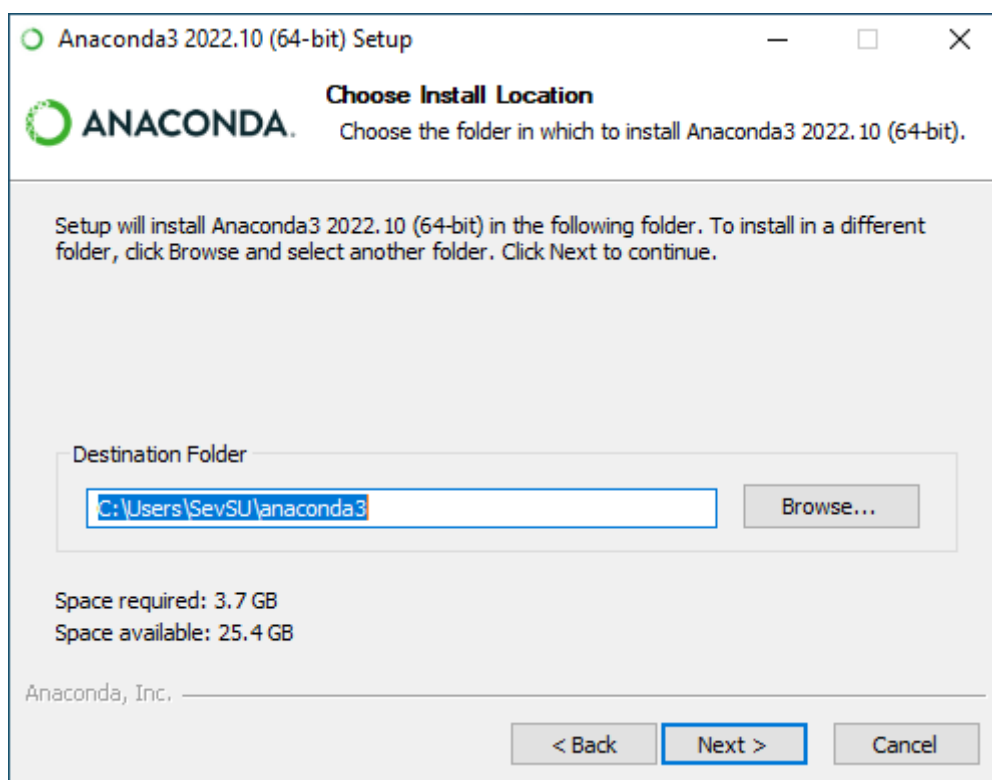


Рис. 11 — Страница задания папки для установки Anaconda

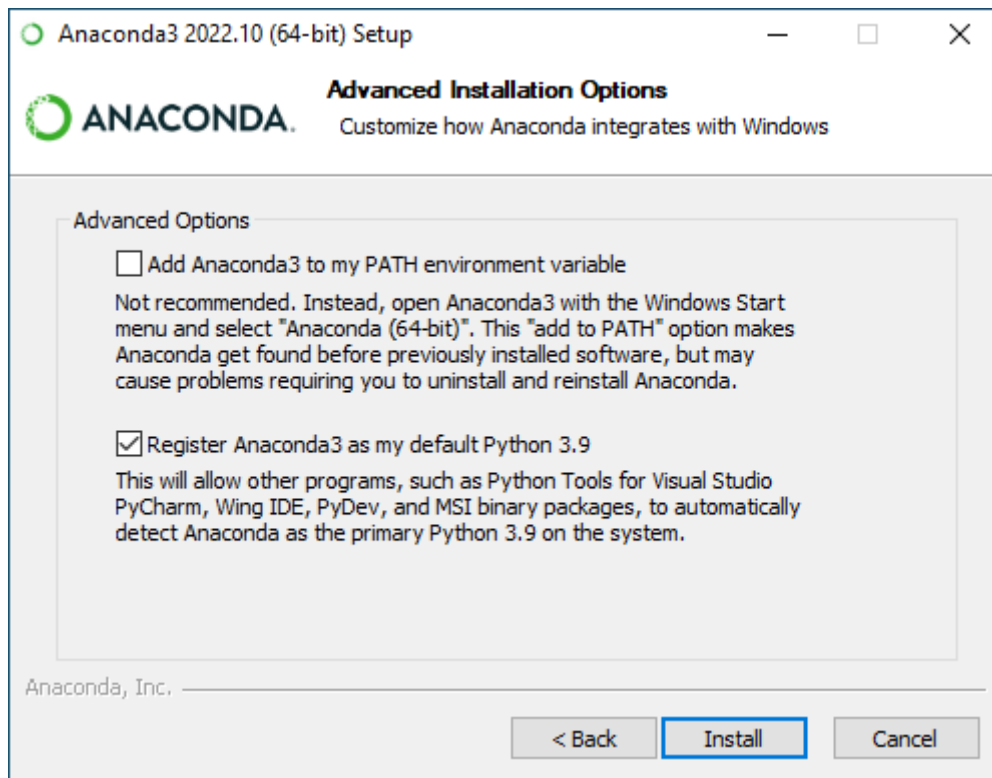


Рис. 12 — Страница конфигурации параметров установки Anaconda

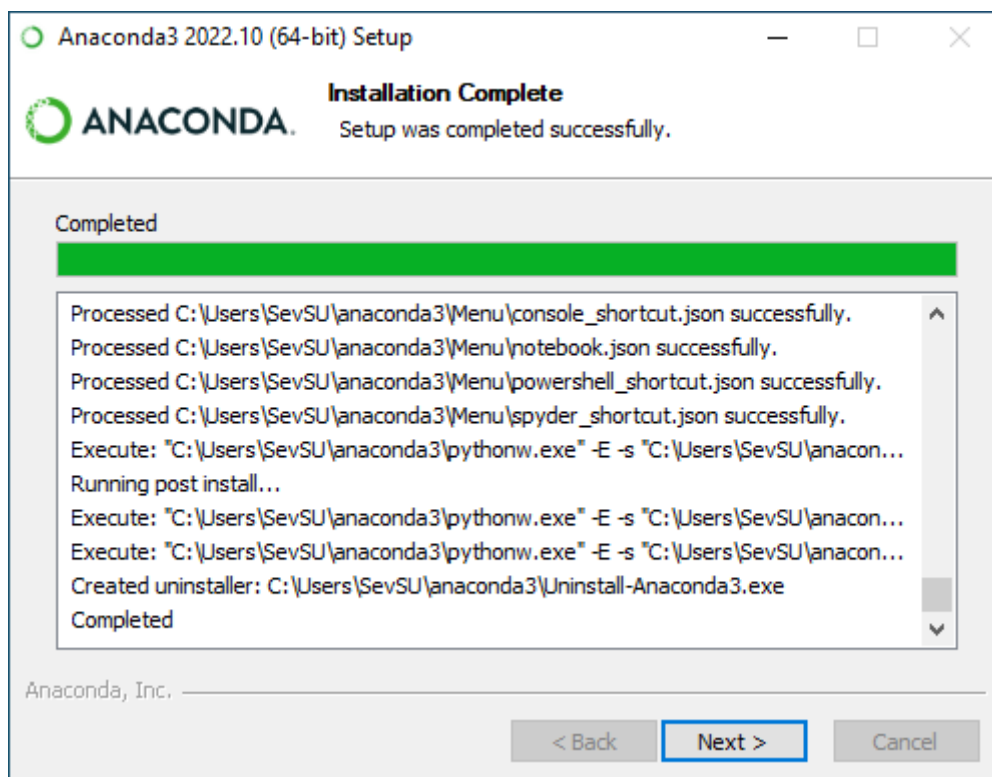


Рис. 13 — Страница прогресса установки Anaconda

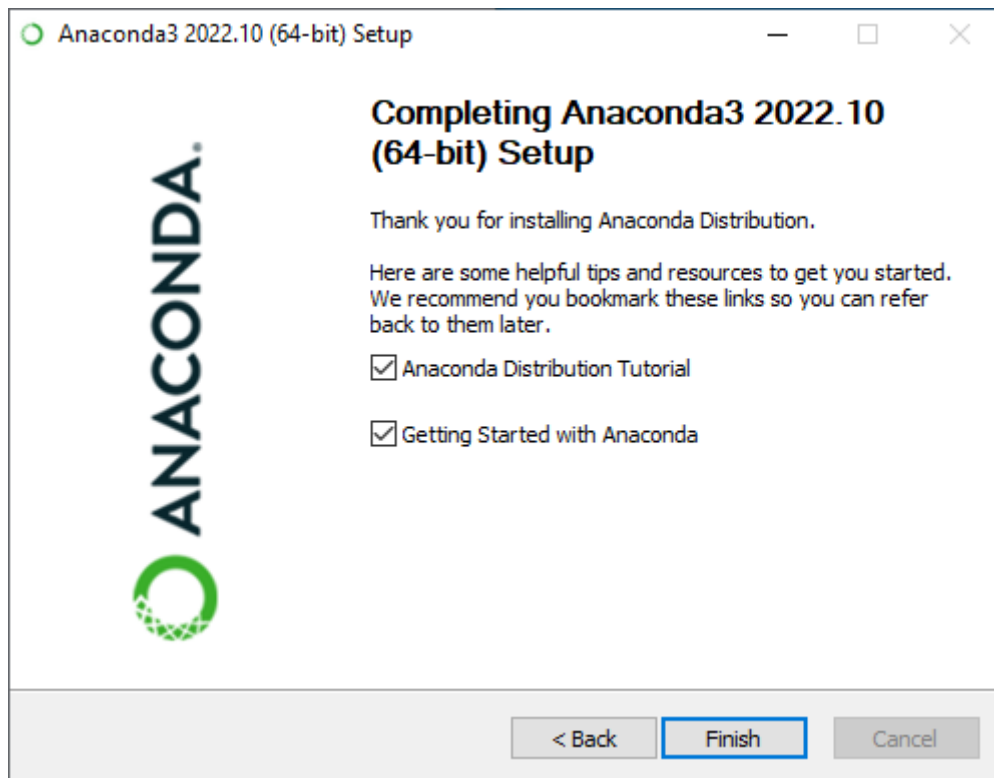


Рис. 14 — Завершающая страница программы-установщика Anaconda

Шаг 3. Для запуска менеджера Anaconda можно воспользоваться системным меню ОС Windows

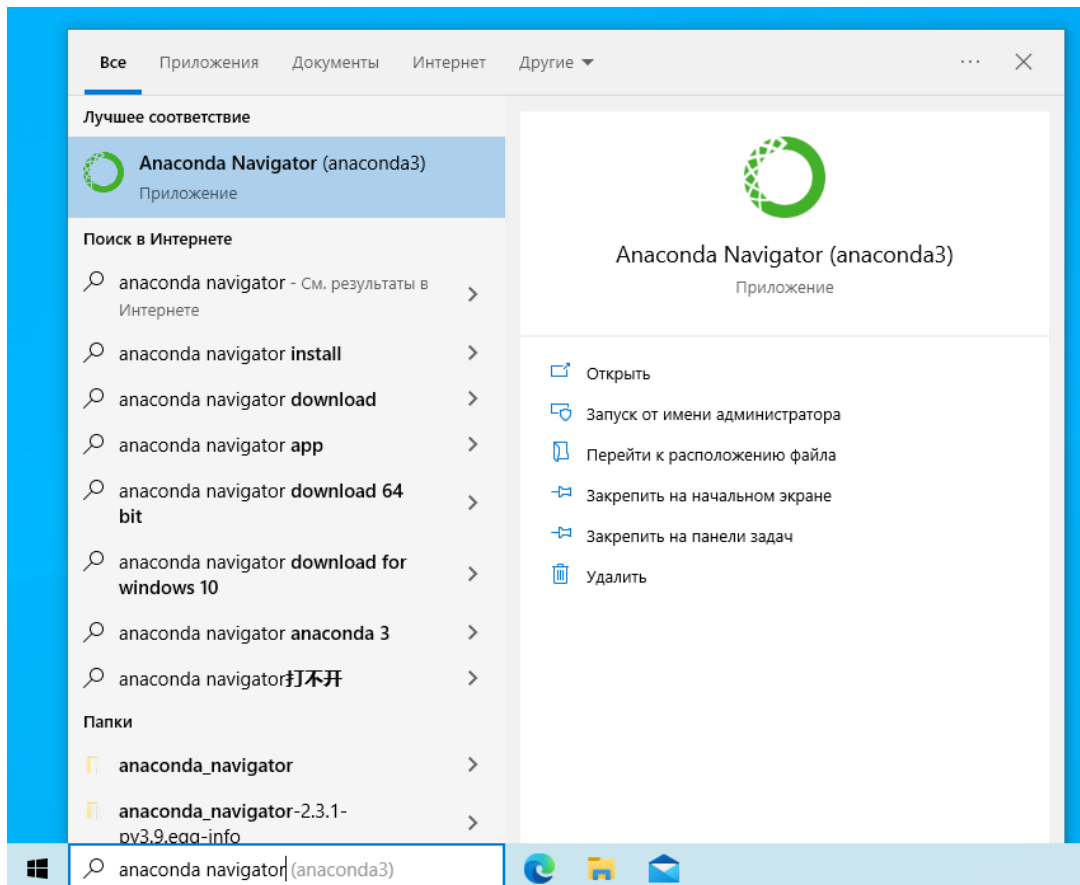


Рис. 15 — Запуск менеджера Anaconda Navigator из системного меню Windows

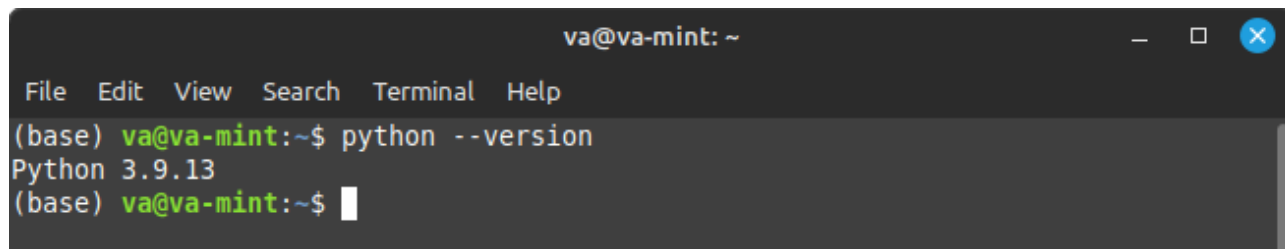
Установка Python, PyCharm

Инструкция по установке для ОС Linux

Для дистрибутива Linux Mint Python входит в набор предустановленного программного обеспечения. Для того чтобы узнать версию установленного Python нужно запустить окно терминала и ввести команду

```
python --version
```

В окне терминала будет выведена информация о текущей версии как это показано на рис. 16.



```
va@va-mint: ~  
File Edit View Search Terminal Help  
(base) va@va-mint:~$ python --version  
Python 3.9.13  
(base) va@va-mint:~$
```

Рис. 16 — Проверка версии Python

Для установки Python на другие дистрибутивы Linux рекомендуется обратиться к справочной системе на сайте разработчика ОС.

Установка IDE PyCharm может быть выполнена несколькими способами. Самый простой заключается в установке с помощью встроенного менеджера программ (Software Manager). Для этого необходимо запустить Software Manager и в строке поиска ввести PyCharm. Если соответствующий пакет будет найден, то дальнейшая установка не вызовет никаких затруднений. Если пакет отсутствует, то необходимо выполнить дополнительные действия.

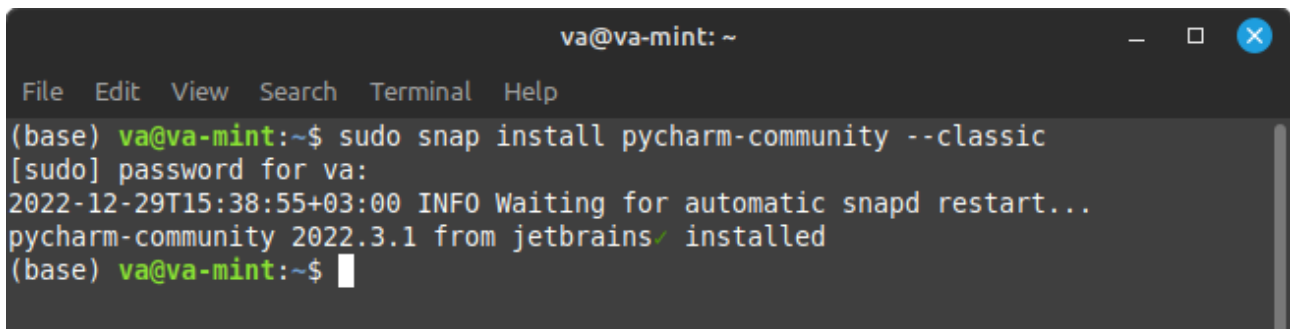
Шаг 1. Включить поддержку установки snap пакетов. Для этого выполнить последовательность команд, приведённую ниже. Последняя команда выполнит перезагрузку системы после завершения конфигурации системных настроек.

```
sudo mv /etc/apt/preferences.d/nosnap.pref  
~/Documents/nosnap.backup  
sudo apt update  
sudo apt install snapd  
sudo reboot
```

Шаг 2. После перезагрузки системы выполнить команду и дождаться окончания установки (Рис. 8)

```
sudo snap install pycharm-community --classic
```

Внимание: устанавливать необходимо PyCharm Community Edition, поскольку данная версия IDE является бесплатной и не требует лицензирования для использования.



```
va@va-mint: ~  
File Edit View Search Terminal Help  
(base) va@va-mint:~$ sudo snap install pycharm-community --classic  
[sudo] password for va:  
2022-12-29T15:38:55+03:00 INFO Waiting for automatic snapd restart...  
pycharm-community 2022.3.1 from jetbrains✓ installed  
(base) va@va-mint:~$
```

Рис. 17 — Установка PyCharm с помощью snap пакета

После окончания установки PyCharm будет доступен через стандартное меню операционной системы (Рис. 9)

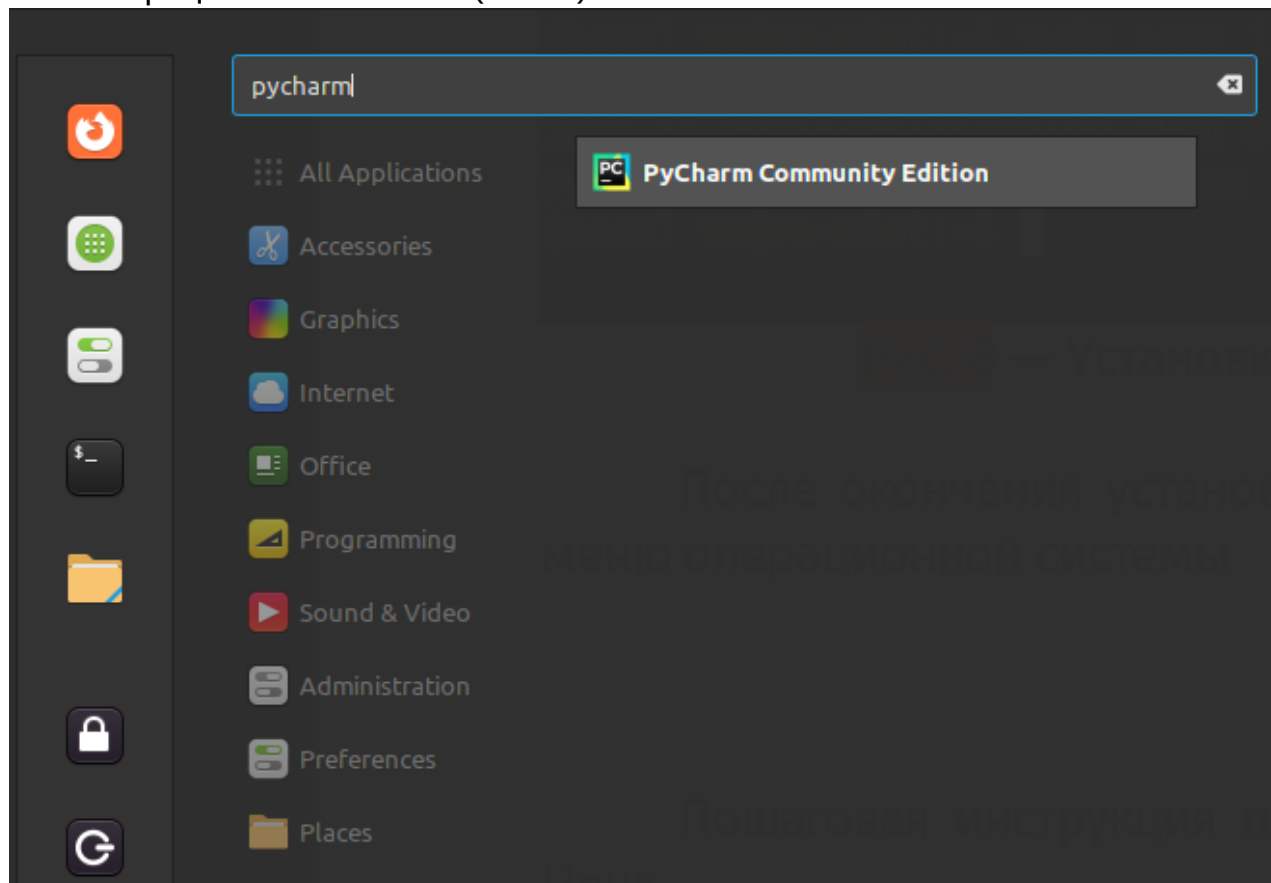


Рис. 18 — Результат установки PyCharm

Инструкция по установке для ОС Windows

Установочный пакет для актуальной версии языка Python доступен по ссылке <https://www.python.org/downloads/>

Шаг 1. Скачать установочный пакет, нажав на кнопку **Download** на странице загрузки

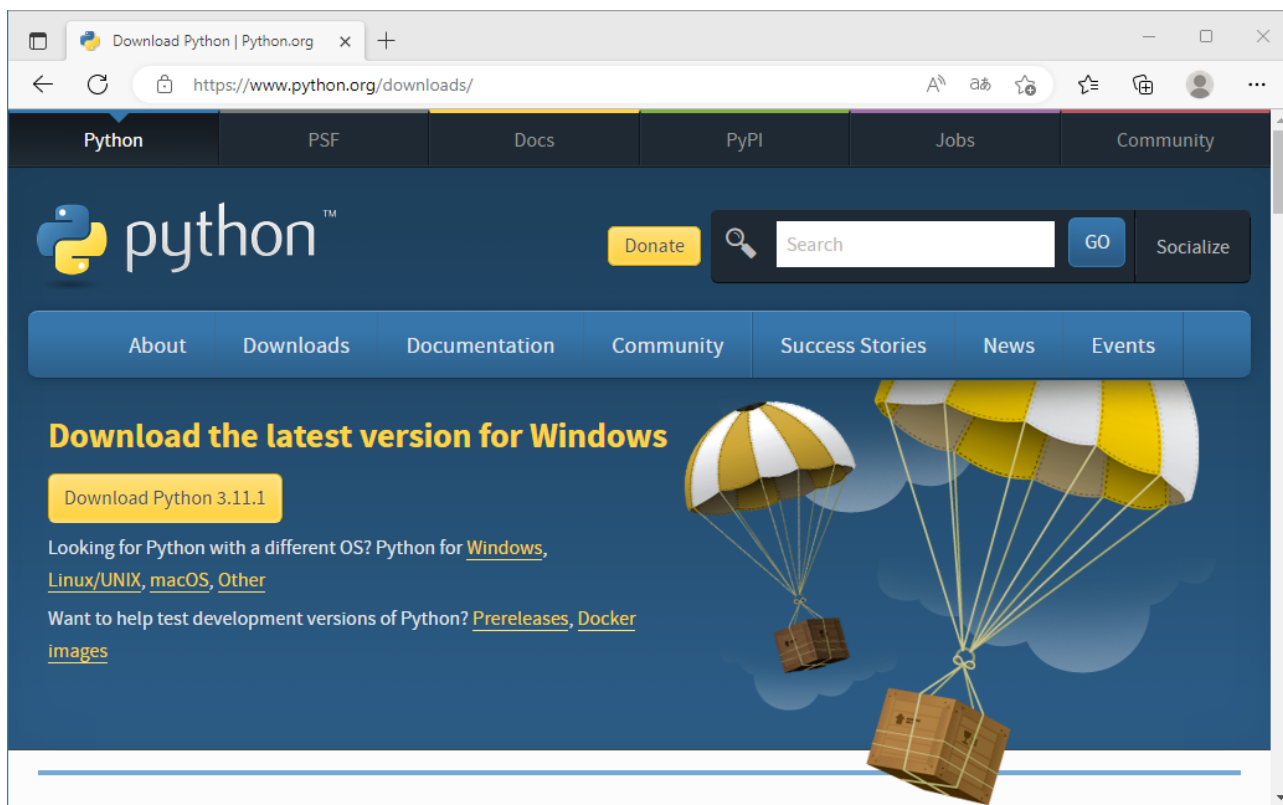


Рис. 19 — Страница загрузки установочного пакета Python для ОС Windows

Шаг 2. По окончании загрузки запустить файл **python-3.11.1-amd64.exe** на исполнение.

Внимание: в зависимости от версии пакета имя скачанного файла может отличаться.

В появившемся окне программы установщика отметить опцию **Add python.exe to PATH** для добавления пути к Python в системную переменную, содержащую список директорий, в которых ОС будет искать исполняемый файл при вызове команды из консоли. Нажать кнопку **Install Now**.

Далее следовать указаниям программы-установщика.

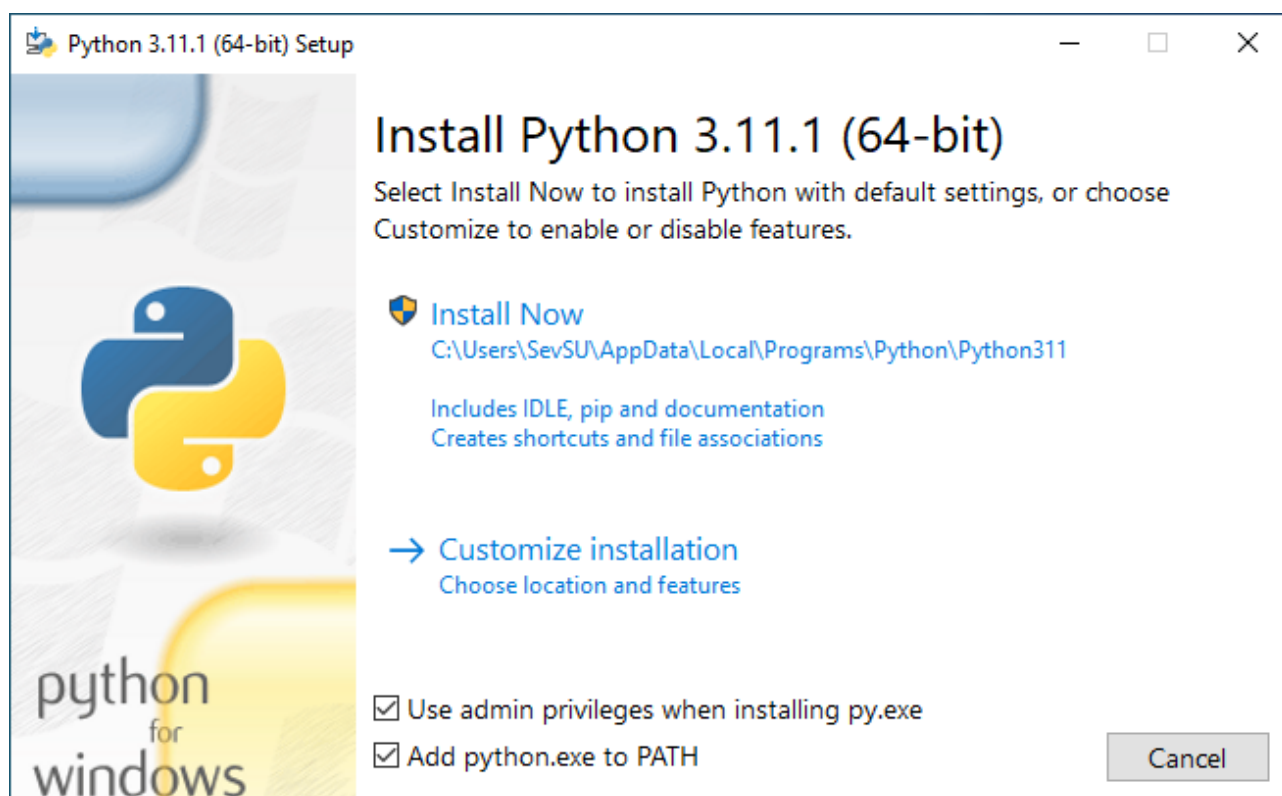


Рис. 20 — Начальная страница программы установщика Python

Шаг 3. Дождаться окончания процесса установки. На последнем шаге установки, нажать на кнопку **Disable path length limit** после чего окно установщика можно закрыть.

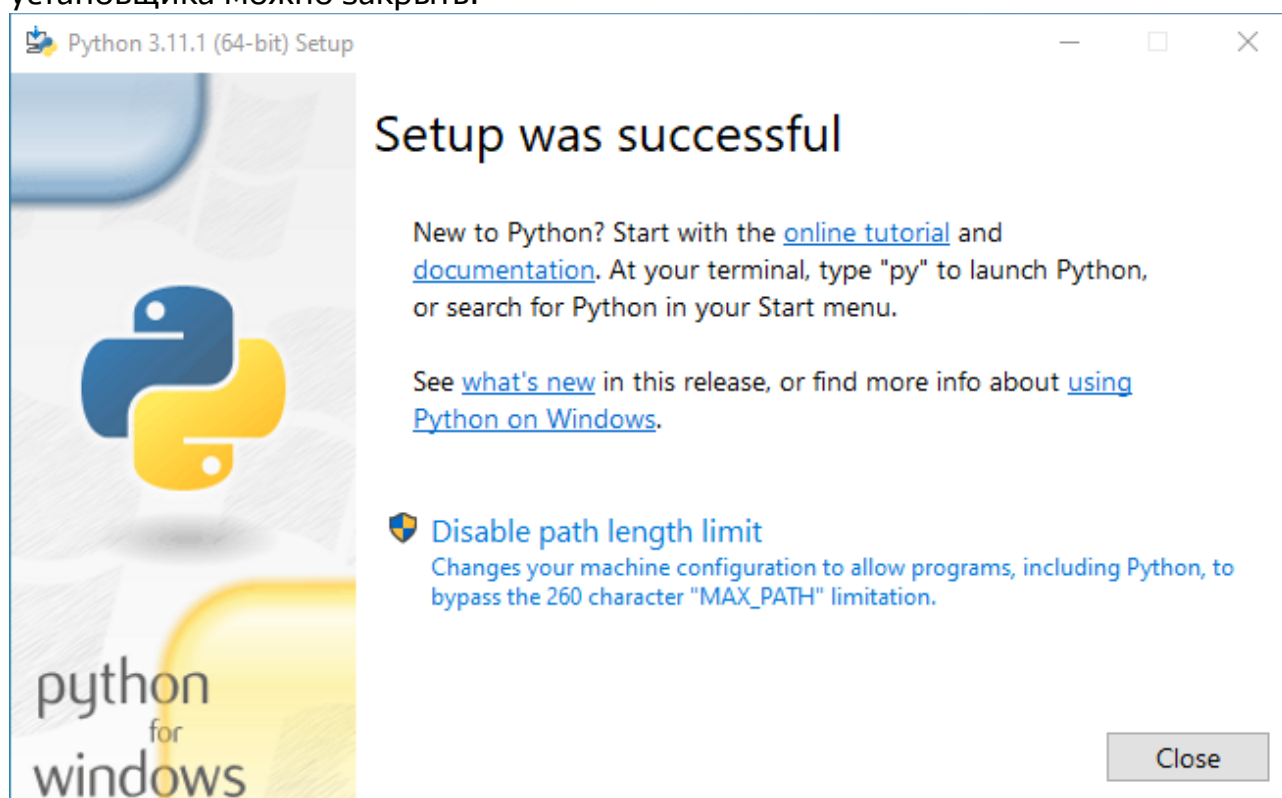
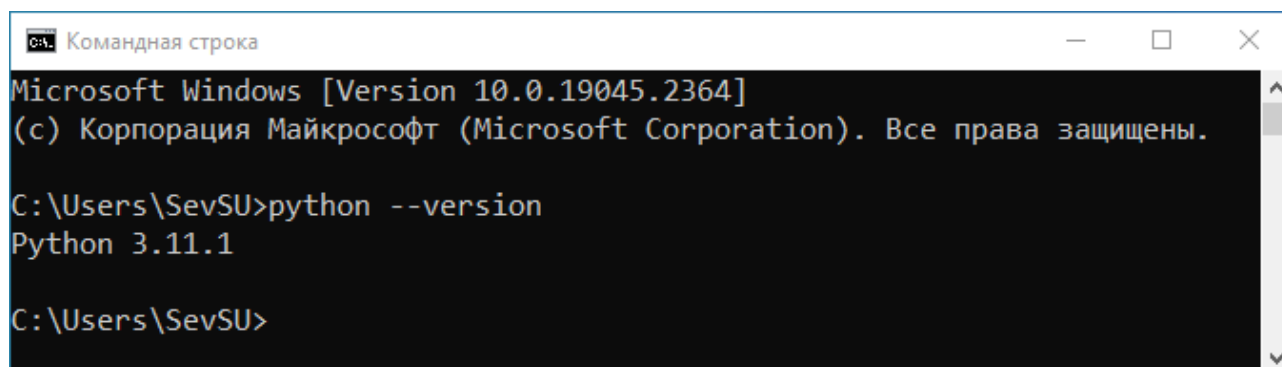


Рис. 21 — Завершающая страницы программы-установщика Python

Шаг 4. Проверить установленную версию Python можно с помощью командной строки Windows открыв окно терминала и введя команду `python --version`



```
Командная строка
Microsoft Windows [Version 10.0.19045.2364]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\SevSU>python --version
Python 3.11.1

C:\Users\SevSU>
```

Рис. 22 — Проверка версии Python

Для установки IDE PyCharm необходимо посетить страницу установки <https://www.jetbrains.com/pycharm/download/#section=windows> и загрузить актуальную версию программы-установщика.

Шаг 1. Скачать установочный пакет, нажав на кнопку **Download** на странице загрузки. Скачивать рекомендуется Community версию, поскольку данная версия продукта является бесплатной и не требует процедуры лицензирования.

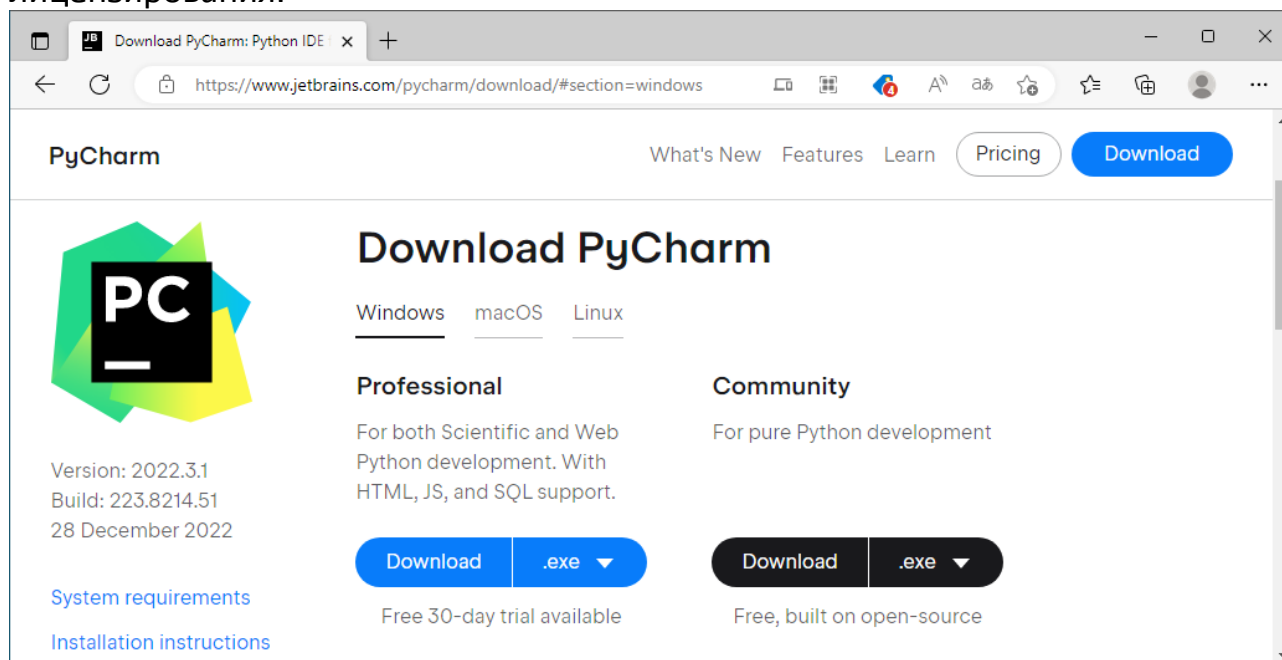


Рис. 23 — Страница загрузки PyCharm

Шаг 23. По окончании загрузки запустить файл `pycharm-community-2022.3.1.exe` на исполнение.

Внимание: в зависимости от версии пакета имя скачанного файла может отличаться.

В появившемся окне программы-установщика

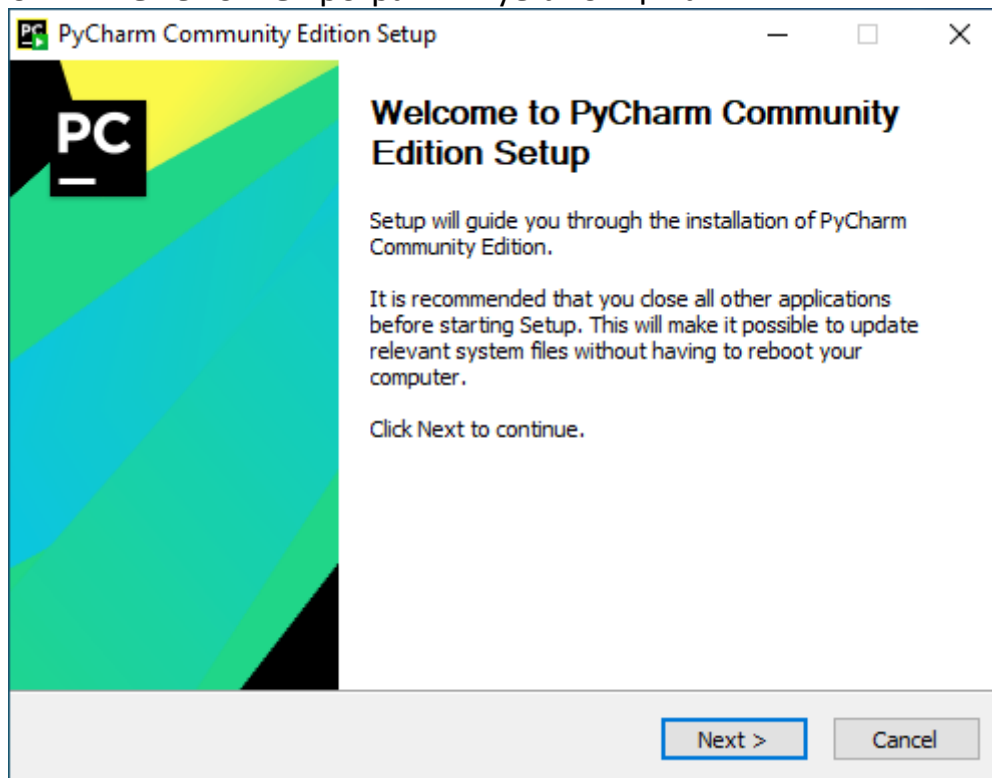


Рис. 24 — Стартовое окно программ-установщика PyCharm

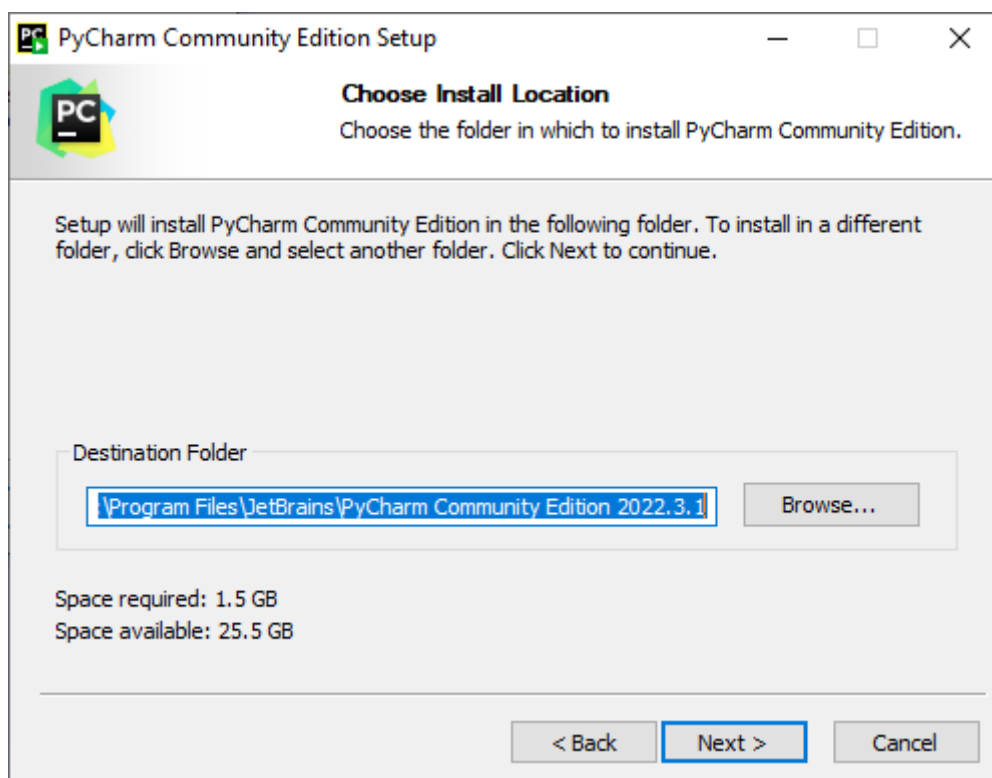


Рис. 25 — Выбор папки установки PyCharm

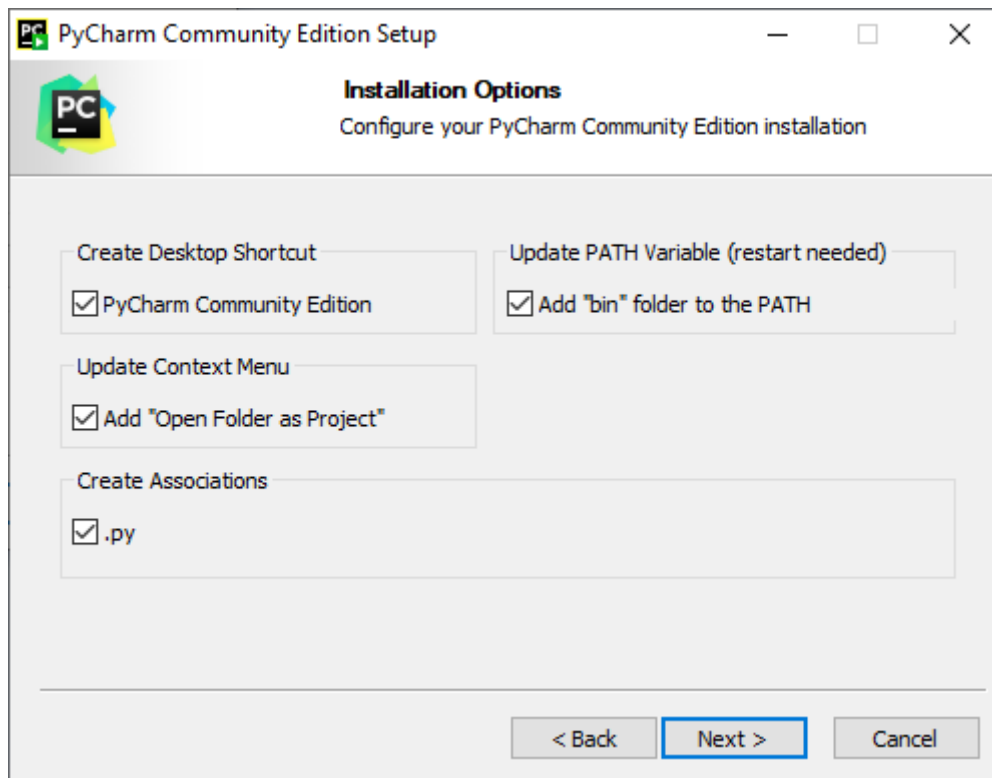


Рис. 26 — Окно настройки установщика PyCharm

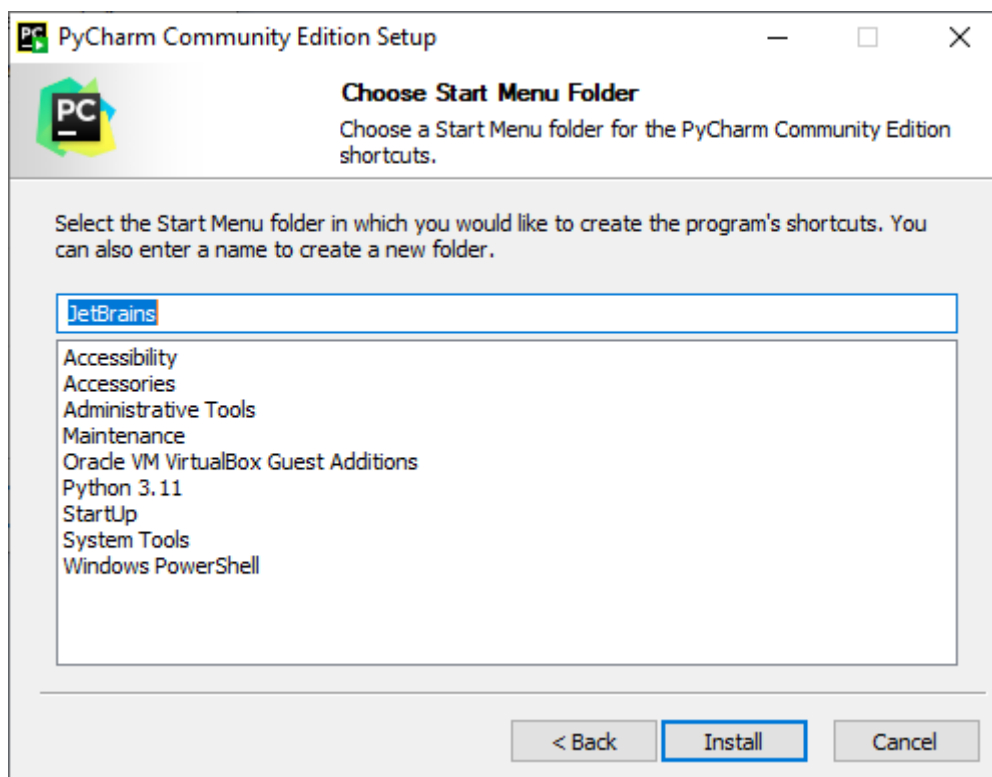


Рис. 27 — Задание имени программы в меню Пуск

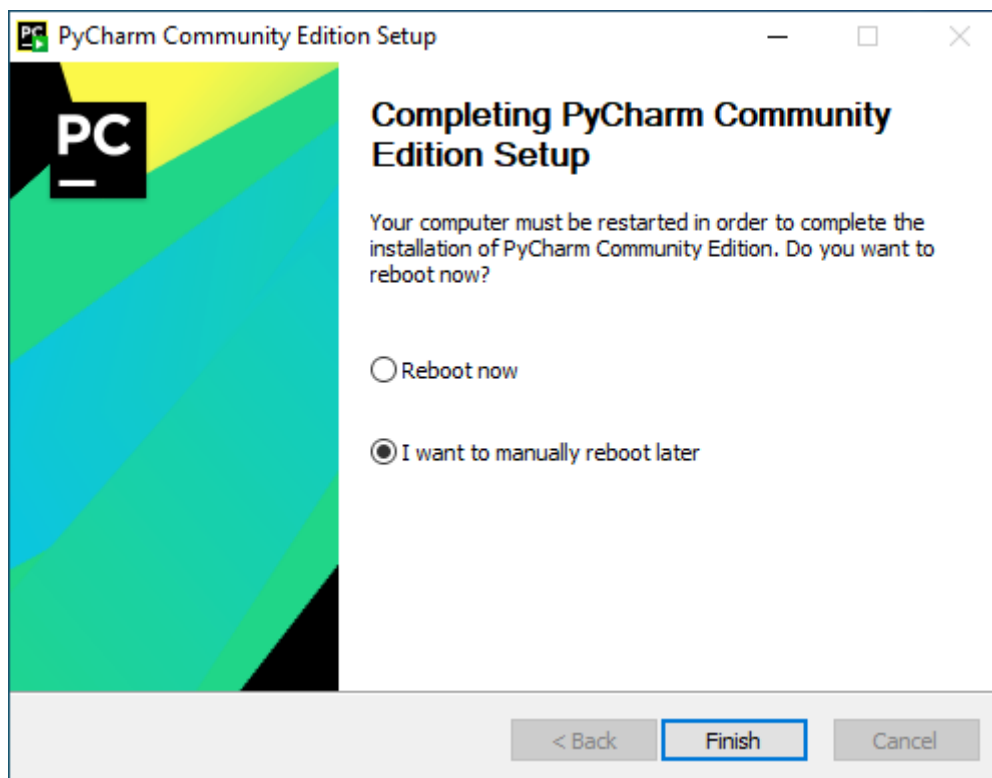


Рис. 28 — Окончание установки PyCharm

После завершения установки рекомендуется выполнить перезагрузку компьютера.

Работа с блокнотом Jupyter Notebook

Работа со скриптами на языке Python возможна в нескольких режимах, среди которых создание текста программ в обычном текстовом редакторе и последующий запуск через командную строку, или использование оболочки IPython (сокращение от «интерактивный Python») в качестве продвинутого интерпретатора Python. Если язык Python представляет собой механизм решения практически любой задачи в области науки о данных, то оболочку IPython можно рассматривать как интерактивную панель управления. Использование оболочки IPython тесно связано с проектом Jupyter, представляющим своеобразный блокнот (текстовый редактор) для браузера, удобный для разработки, совместной работы и использованию ресурсов, а также для публикации научных результатов.

Блокнот Jupyter представляет собой веб-приложение для командной оболочки IPython, позволяющее создавать скрипты на языке Python и предоставляющее большой набор возможностей динамической визуализации. Помимо выполнения операторов, блокнот Jupyter позволяет пользователю вставлять форматированный текст в блокнот, добавлять статические и динамические визуализации, математические формулы, виджеты JavaScript и

многое другое. Блокноты Jupyter могут быть сохранены таким образом, чтобы другие пользователи могли открывать их в своих системах.

Блокнот Jupyter должен иметь возможность взаимодействовать с основным ядром системы. Для этого предназначены специальные протоколы интерактивных вычислений. В ядре Jupyter для Python в качестве такого ядра используется IPython.

Запуск Jupyter Notebook проще всего выполнить из панели Anaconda. Для этого нужно запустить Anaconda Navigator, после чего нажать кнопку Launch, как это показано на рис. 6.

На большинстве платформ Jupyter Notebook открывается в окне браузера, установленного в системе по умолчанию. На рис. 29 показано как выглядит окно начального запуска, в котором можно выбрать блокнот для запуска или создать новый блокнот.

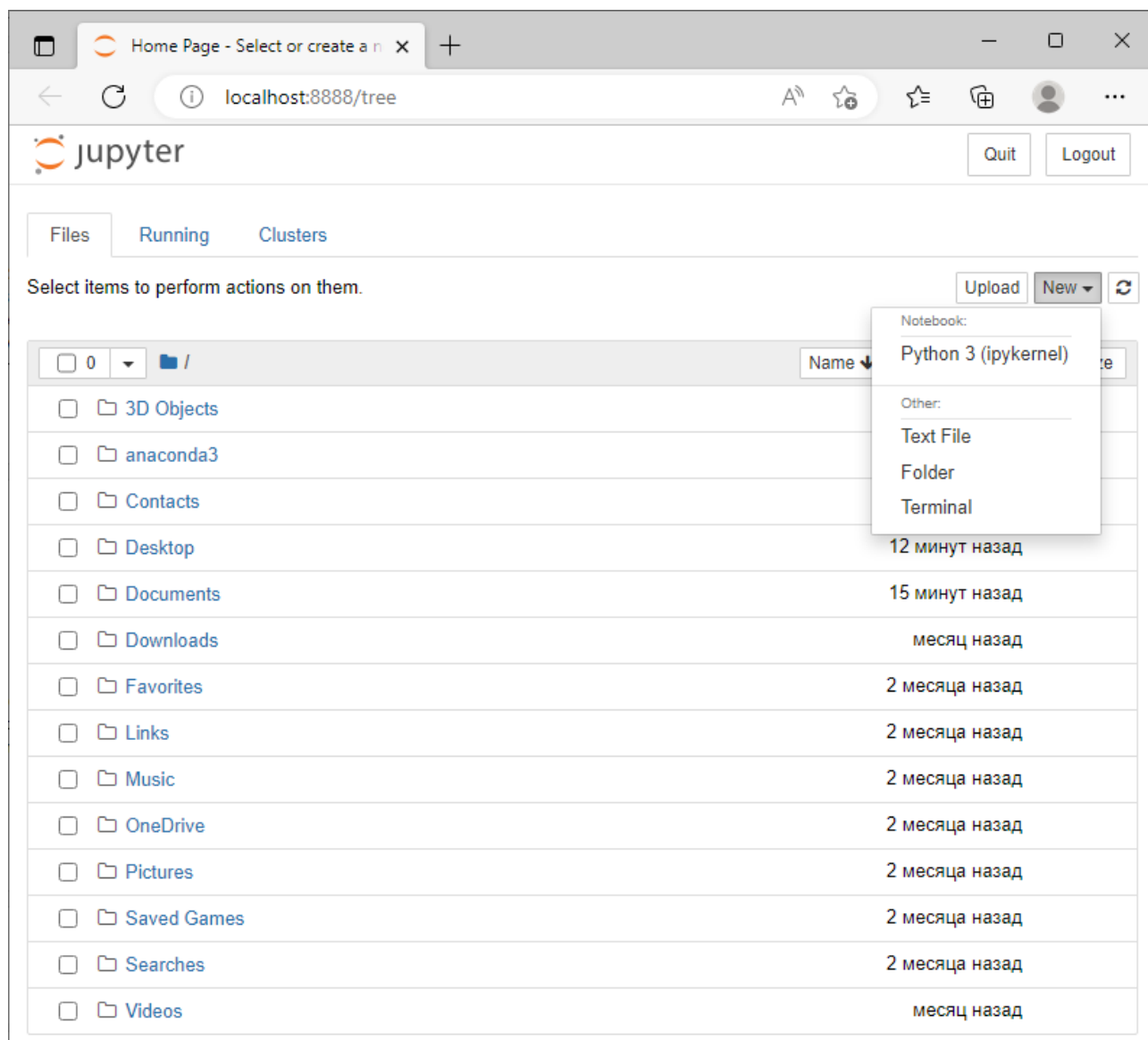


Рис. 29 — Окно начального запуска Jupyter Notebook

Для создания нового блокнота необходимо в комбобоксе **New** выбрать опцию **Python 3 (ipykernel)** в результате чего откроется новое окно браузера с новым блокнотом, как это показано на рис. 30.

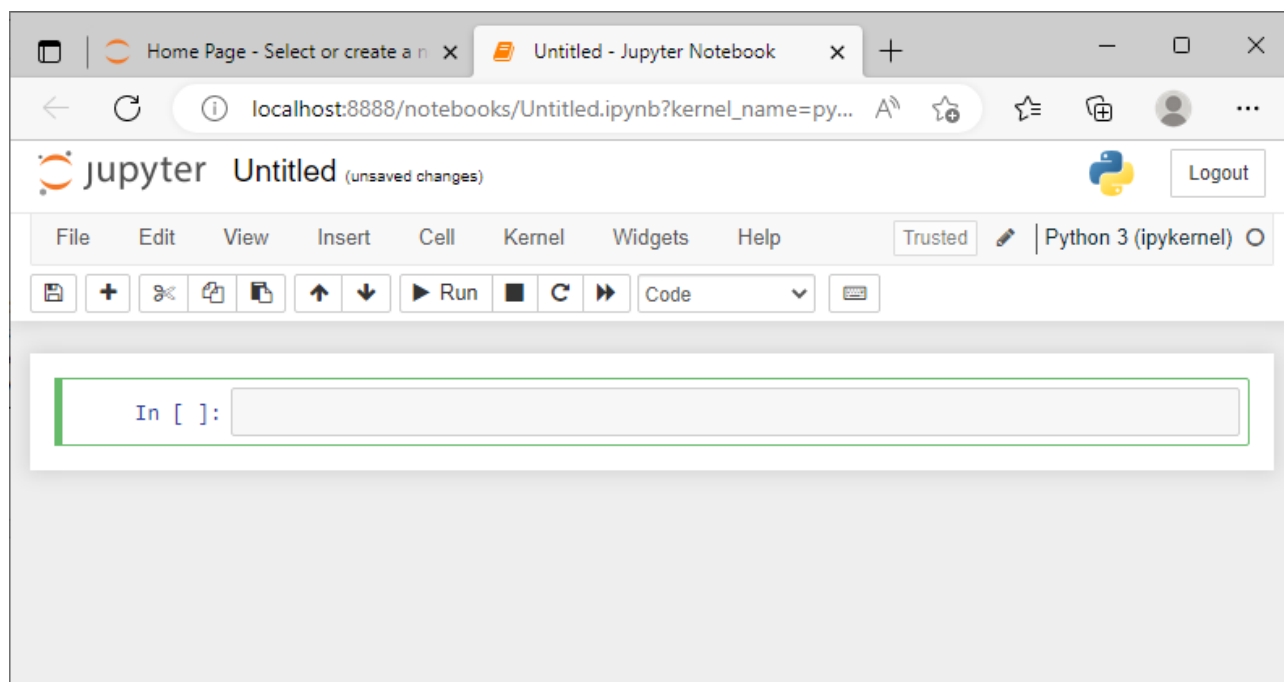


Рис. 30 — Окно с новым блокнотом Jupyter Notebook

Вновь созданный блокнот рекомендуется переименовать. Для этого нужно отредактировать имя по умолчанию **Untitled** в верхней части окна, слева от логотипа Jupyter. Под логотипом находятся пункты меню блокнота и панель инструментов, предназначенные для выполнения манипуляций с ячейками блокнота. Ниже панели инструментов располагается рабочая область с ячейками, предназначенные для ввода команд (ячейки In[]) и вывода результатов (ячейки Out[]). Блокноты Jupyter сохраняются с расширением *.ipynb* и располагаются в каталоге *C:\Users\[Имя_пользователя]*.

Для начала ввода команды, необходимо кликнуть по ячейке, после чего ввести команду или последовательность команд. Для запуска содержимого ячейки на выполнение необходимо нажать комбинацию клавиш **Shift + Enter**.

В результате, если код ячейки не содержит ошибок, то он будет выполнен и автоматически создается новая ячейка для ввода следующей последовательности команд, так как это показано на рис. 31. При необходимости можно вернуться к предыдущей ячейке и запустить код, находящийся в ячейке, на выполнение еще раз. Если необходимо выполнить запуск на выполнение всех ячеек, входящих в блокнот, то для этого можно воспользоваться пунктом меню **Kernel** → **Restart & Run All**. В этом случае осуществляется перезапуск ядра и ячейки блокнота выполняются по порядку, начиная с первой.

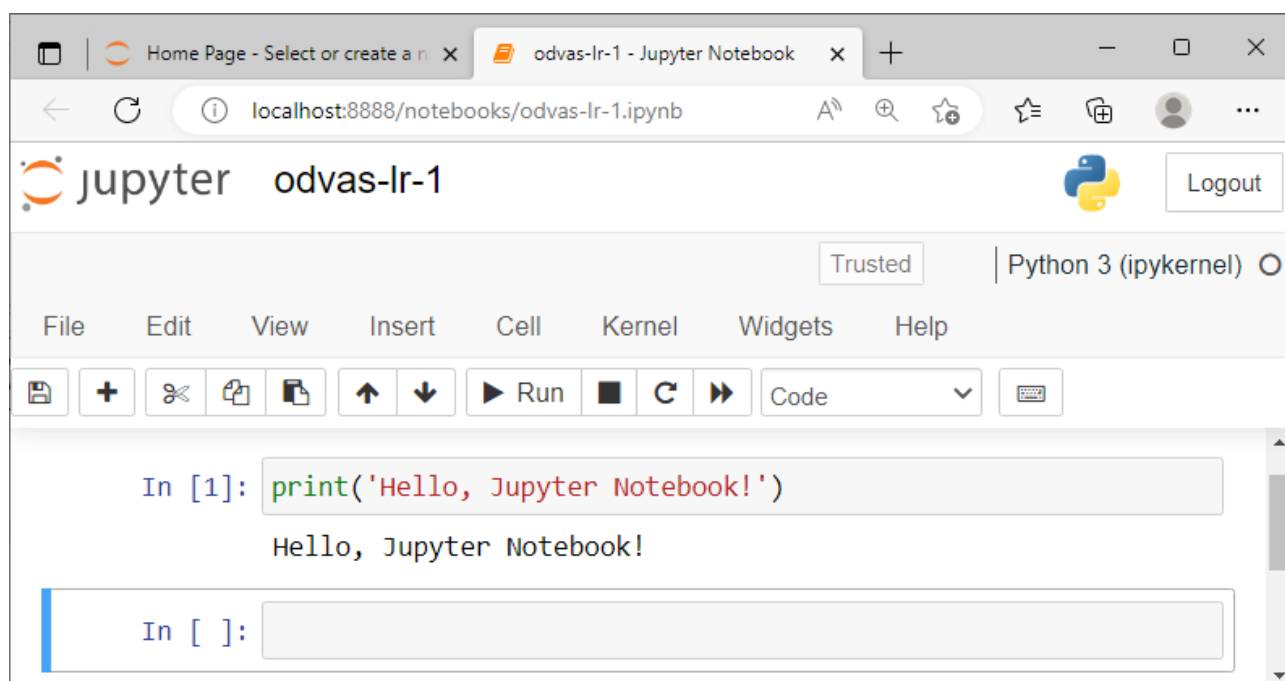


Рис. 31 — Ввод команды в ячейку Jupyter Notebook

Как видно на рис. 31, вновь созданная ячейка обведена серым прямоугольником с левой границей окрашенной в синий цвет. Это означает, что данная ячейка выделена, и с ней можно выполнять определенные манипуляции. Например, данную ячейку можно удалить вместе с ее содержимым, используя сочетание горячих клавиш **D + D**. Если граница прямоугольника окрашена в зеленый цвет, это означает, что ячейка находится в режиме редактирования и готова для ввода новой последовательности команд (см. рис. 30). Для выделения ячейки, необходимо выполнить щелчок левой кнопкой мыши слева от надписи **In [N]**, где N – номер интересующей ячейки.

Для работы с блокнотами Jupyter Notebook рекомендуется выучить набор сочетаний горячих клавиш, что в значительной степени ускорит работу с блокнотом. Горячие клавиши могут быть из стандартного набора, а также могут быть назначены пользователем самостоятельно и по своему усмотрению. Для вывода текущих настроек сочетаний горячих клавиш необходимо перейти в пункт меню **Help** → **Keyboard Shortcuts**.

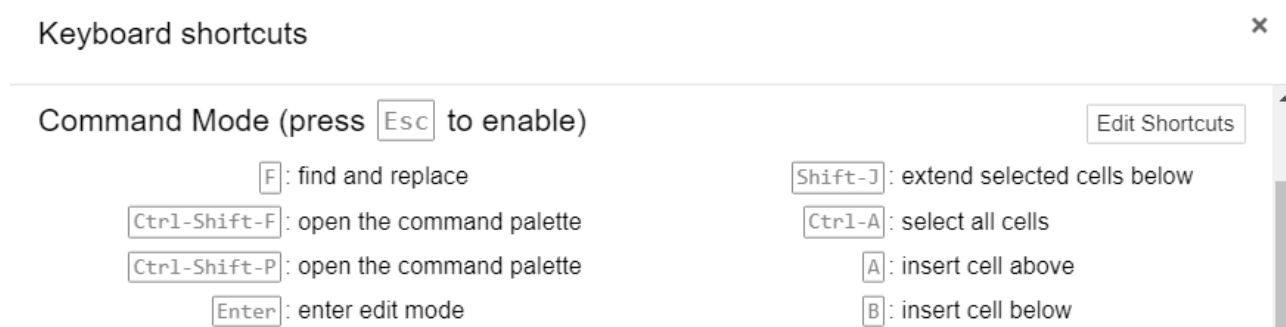



Рис. 32 — Окно с информацией о сочетаниях горячих клавиш

command mode


- switch to edit mode
- h** shows all keyboard shortcuts
- change the selected cell to the above cell
- change the selected cell to the below cell
- a** insert the code cell above
- b** insert the code cell below
- x** cut the selected cell
- d** **d** deletes the selected cell
- c** copy the selected cell
- v** paste the cell below
- + **v**
paste the cell above the current cell
- z** undoes the cell deletion
- p** open the command palette
- y** change the cell type to Code
- m** change the cell type to Markdown
- r** change the cell type to Raw
- Space** scroll notebook down
- + **Space** scroll notebook up

edit mode

- Esc** exit from edit mode to command mode
- code completion
- Ctrl** + **[** decrease indentation
- Ctrl** + **]** increase indentation
- Ctrl** + **a** select all text in the code
- Ctrl** + **z** undo previous action
- Ctrl** + **Home** go to the cell start
- Ctrl** + **End** go to the cell end
- Ctrl** + go one word right
- Ctrl** + go one word left



SHORTCUTS



both modes

- Ctrl** + **s** save the notebook and checkpoint
- Ctrl** + execute current cell
- Alt** + execute current cell and insert a code cell below
- + execute current cell and switch the focus to the cell below

Рис. 33 — Основные сочетания горячих клавиш Jupyter Notebook

Создание проекта в PyCharm

Для создания нового проекта с помощью IDE PyCharm необходимо запустить среду разработки, в появившемся окне нажать на кнопку **New Project** как это показано на рис. 34.

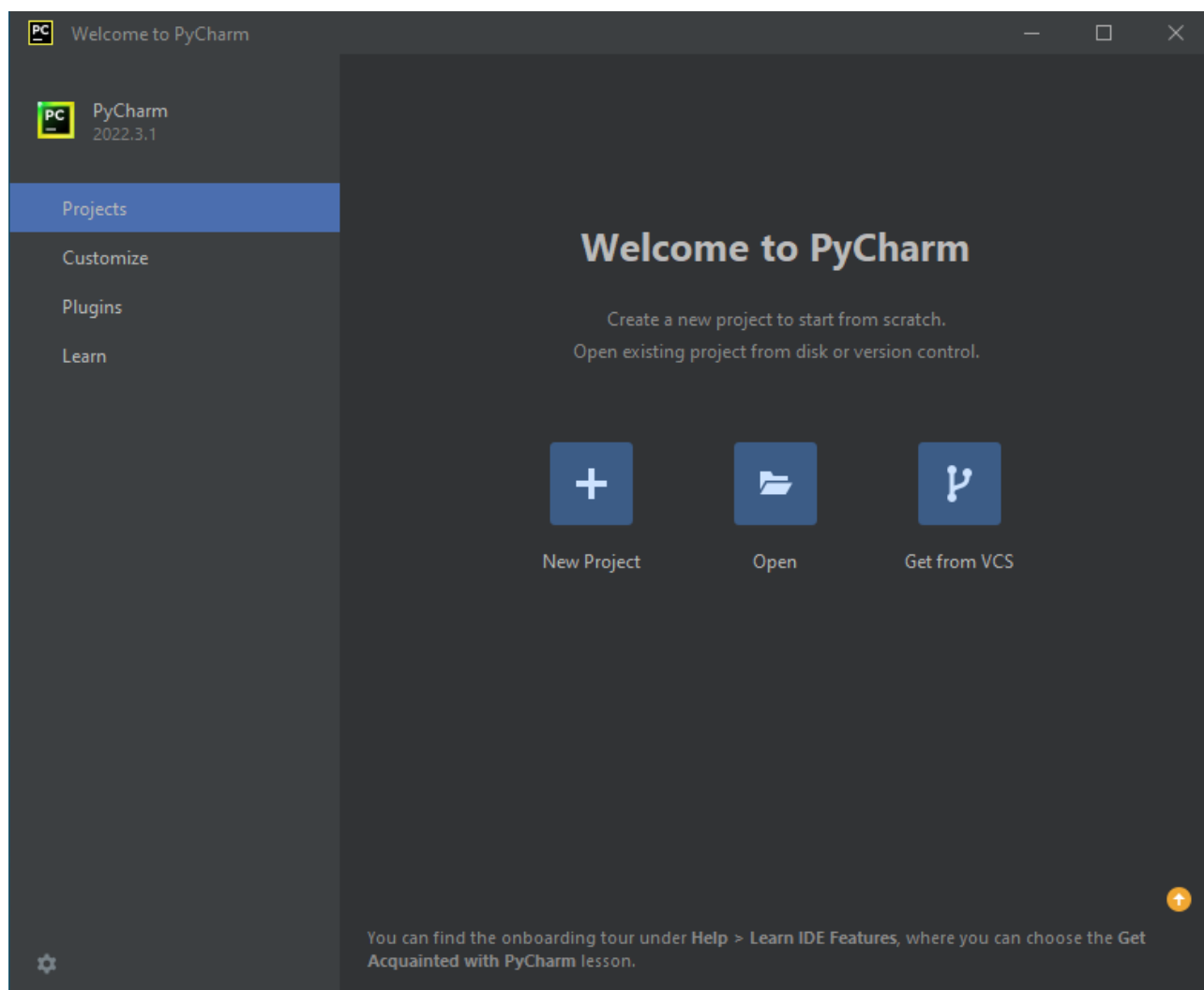


Рис. 34 — Окно приветствия PyCharm

На следующем шаге необходимо выполнить конфигурацию будущего проекта. Окно с настройками проекта представлено на рис. 35. Прежде всего в поле **Location** задаётся путь к папке проекта. В разделе **Python Interpreter** необходимо определить тип окружения, которое будет использоваться для проекта. Это может быть общесистемное окружение (Pipenv) или виртуальное окружение (Virtualenv). Выбор типа окружения определяется тем, из какого хранилища будут подключаться библиотеки. В случае общесистемного окружения все проекты будут использовать одни и те же версии библиотек. Если при создании проекта выбрать виртуальное окружение, тогда проект может использовать свои собственные версии библиотек, независимо от того,

установлена библиотека в общесистемном окружении или нет. Интерпретатор Python при импорте модуля или пакета ищет его сначала в пользовательском окружении, затем — в общесистемном. Перед закрытием окна настроек необходимо убедиться в том, что в поле **Base interpreter** прописан действующий путь к интерпретатору Python.

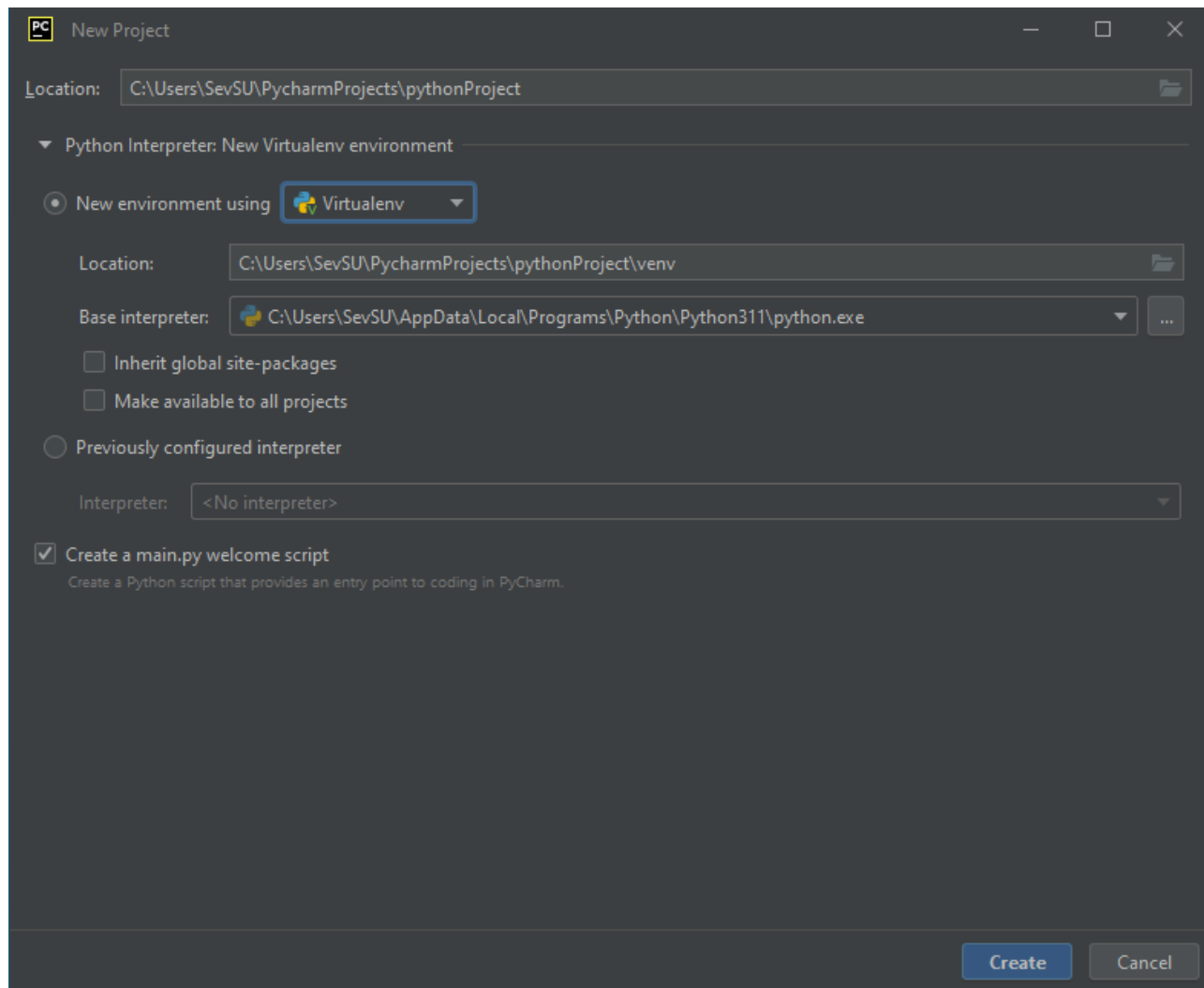


Рис. 35 — Окно настроек проекта PyCharm

После нажатия на кнопку **Create** окна настроек проекта, открывается основное окно разработки, которое представлено на рис. 36.

Окно условно разделено на три области: в левой части окна представлено дерево, отражающее структуру файлов и папок проекта, в правой части расположен редактор кода, внизу находятся вспомогательные закладки для отображения окна терминала, консольного вывода и других системных утилит. Для запуска скрипта на выполнения необходимо нажать на кнопку с зелёной стрелкой в правой части панели инструментов.

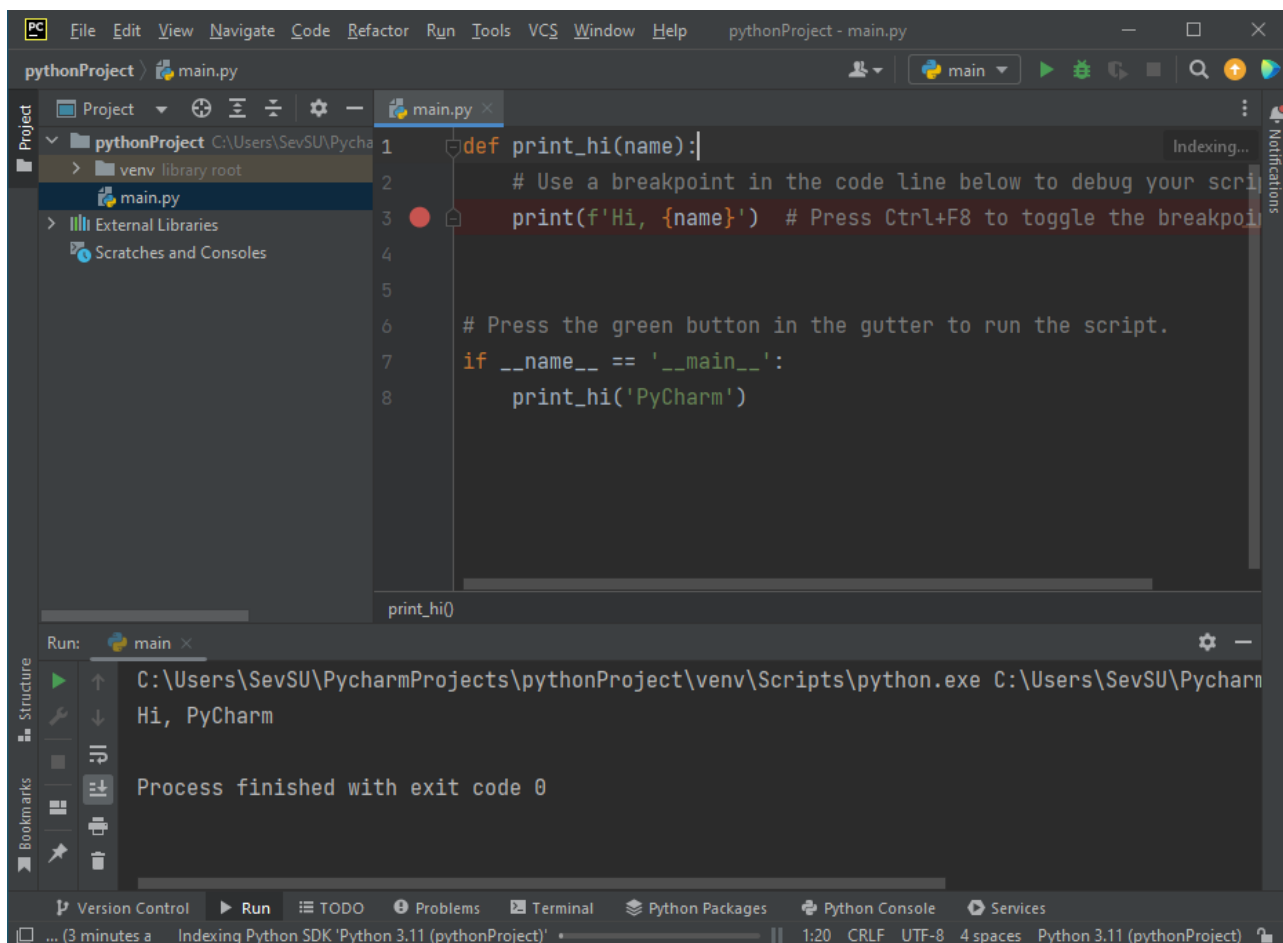


Рис. 36 — Основное окно разработки PyCharm

Поскольку при создании проекта использовано виртуальное окружение, все библиотеки должны быть установлены заново. Для добавления библиотеки необходимо открыть окно настроек (пункт меню **File** → **Settings** или сочетание горячих клавиш **Ctrl+Alt+S**) и в дереве настроек выбрать пункт **Project** → **Python Interpreter** как это показано на рис. 37. В появившемся списке будут перечислены установленные в виртуальное окружение библиотеки и их версии. При необходимости добавить библиотеку, нужно нажать на кнопку **+** и в появившемся окне выбрать необходимую библиотеку. После чего нажать кнопку **Install Package** и дождаться окончания установки. На рис. 38 представлен пример установки библиотеки NumPy, а на рис. 39 пример подключения библиотеки к коду с помощью команды `import` и вывод версии библиотеки в консоль с помощью функции `print`. При необходимости можно указать версию библиотеки, которую необходимо установить, в противном случае будет установлена последняя актуальная версия библиотеки.

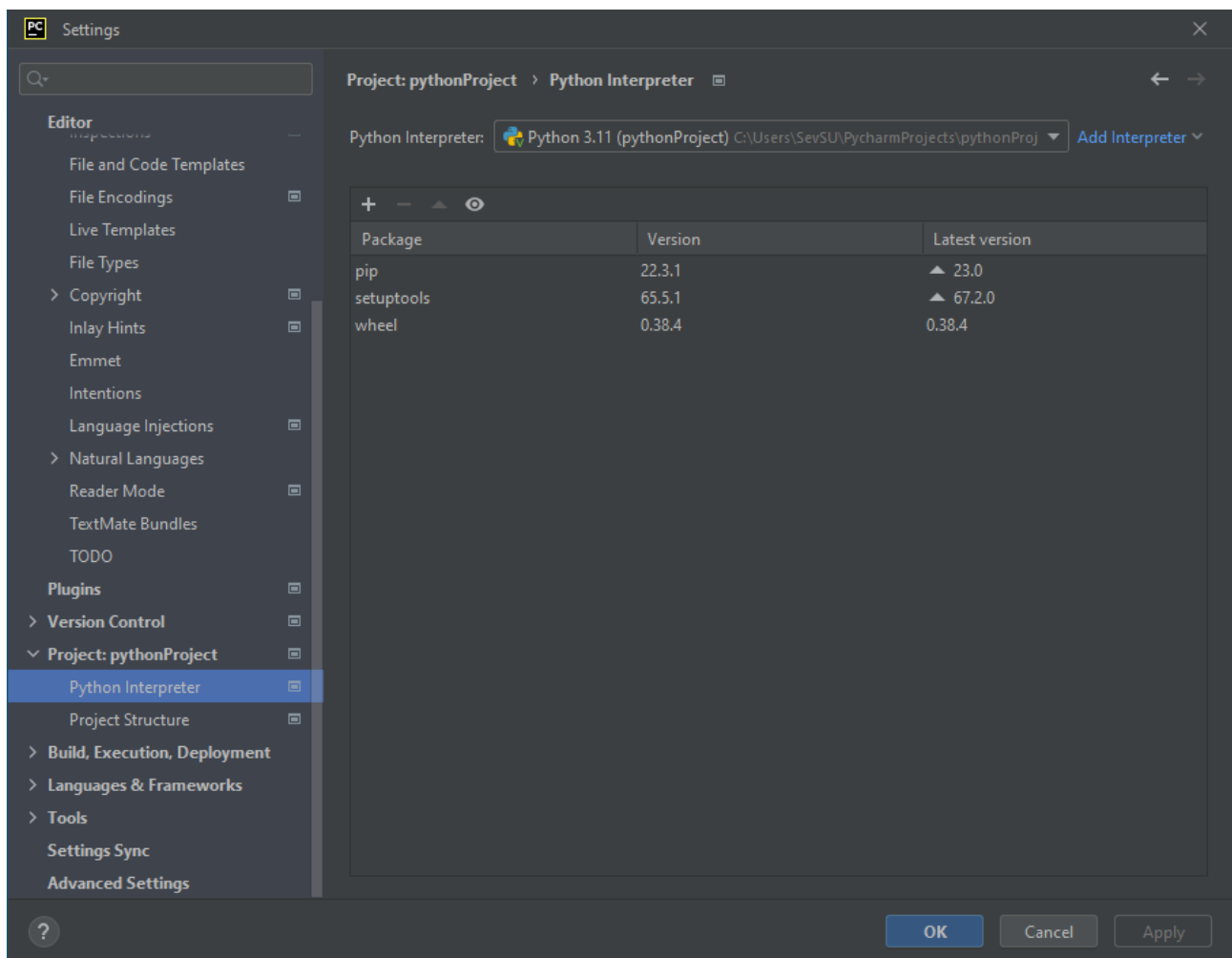


Рис. 37 — Окно настроек проекта

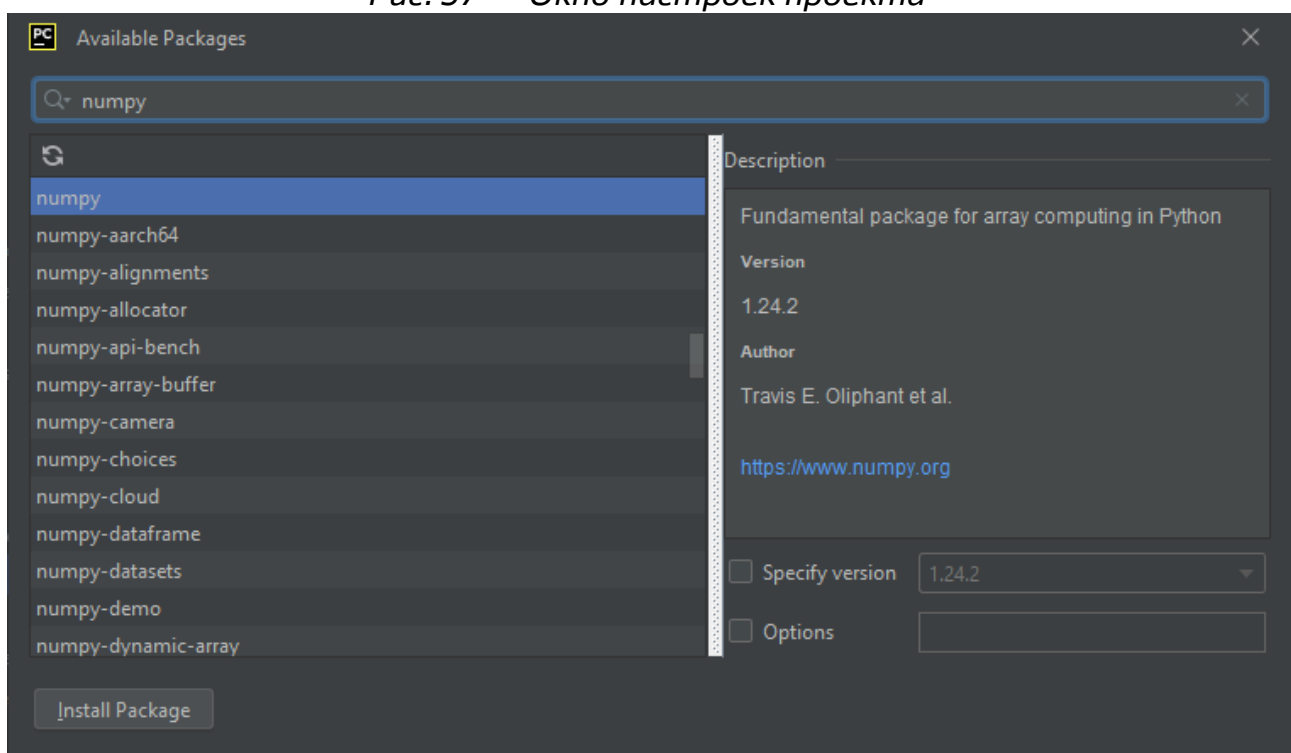


Рис. 38 — Окно установки библиотека

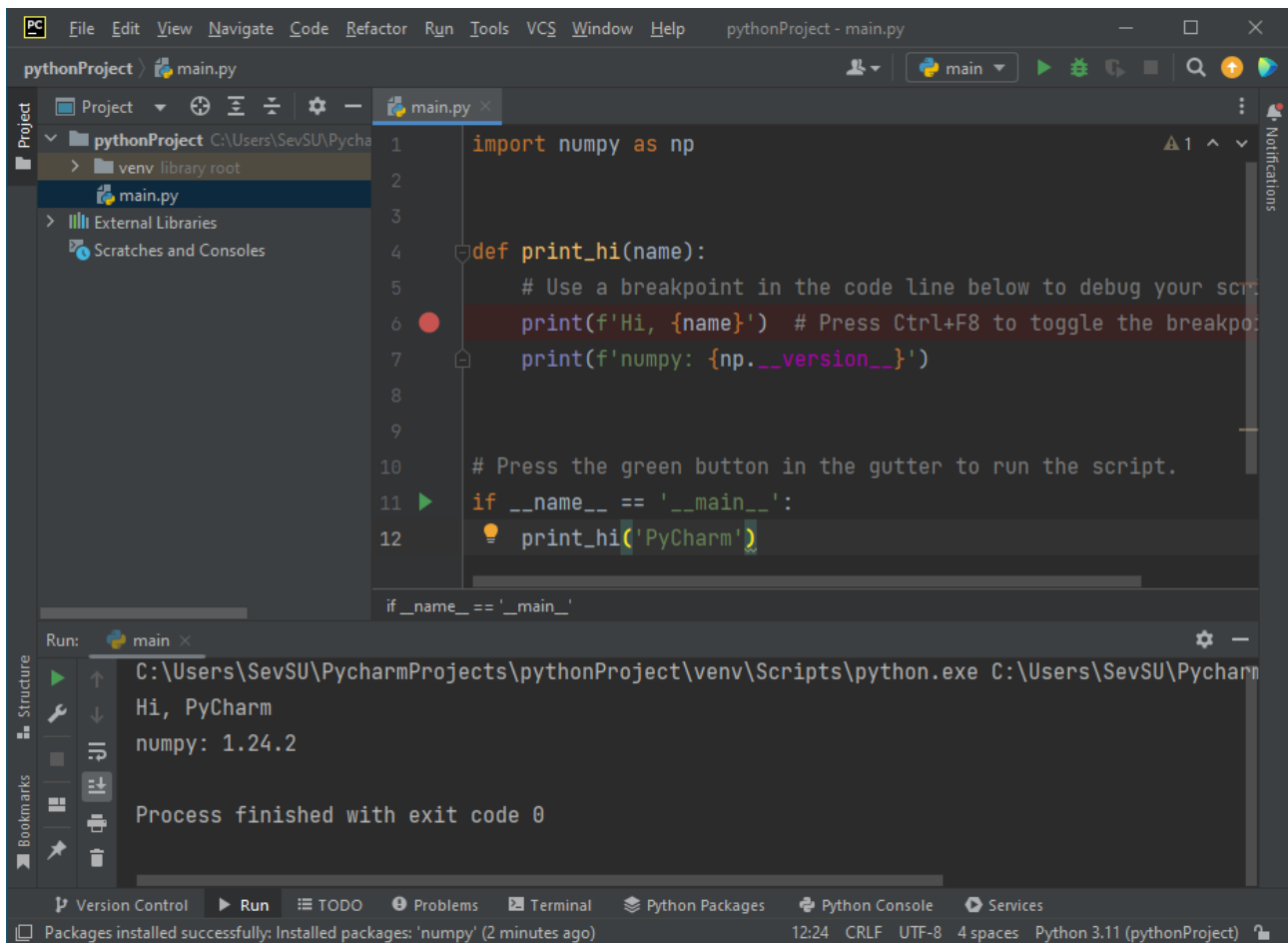


Рис. 39 — Подключение библиотеки и вывод версии библиотеки в консоль

Установка пакетов с помощью менеджера пакетов `pip`

Чистый Python, как язык программирования, достаточно плохо адаптирован для решения задач анализа данных. Чтобы превратить его в мощный инструмент анализа, необходимо использовать специализированные библиотеки, такие как NumPy, Pandas, Matplotlib и др. В случае, если разработка ведётся в пакете Anaconda, то пользователю не придётся выполнять никаких дополнительных действий по конфигурации окружения, но если используется связка Python + PyCharm, об установке пакетов придётся позаботиться самостоятельно.

Установка пакетов Python осуществляется с помощью менеджера пакетов `pip`, входящего в стандартный пакет установки Python. Работа с `pip` осуществляется в режиме командной строки. Основной формат команды для манипулирования с пакетом имеет вид

`pip [команда] [имя_библиотеки]`

Основные команды менеджера пакетов `pip` приведены в табл. 1. Полное описание формата команд для различных операционных систем можно найти по ссылке <https://pip.pypa.io/en/stable/getting-started/>

Табл. 1 — Основные команды менеджера пакетов *pip*

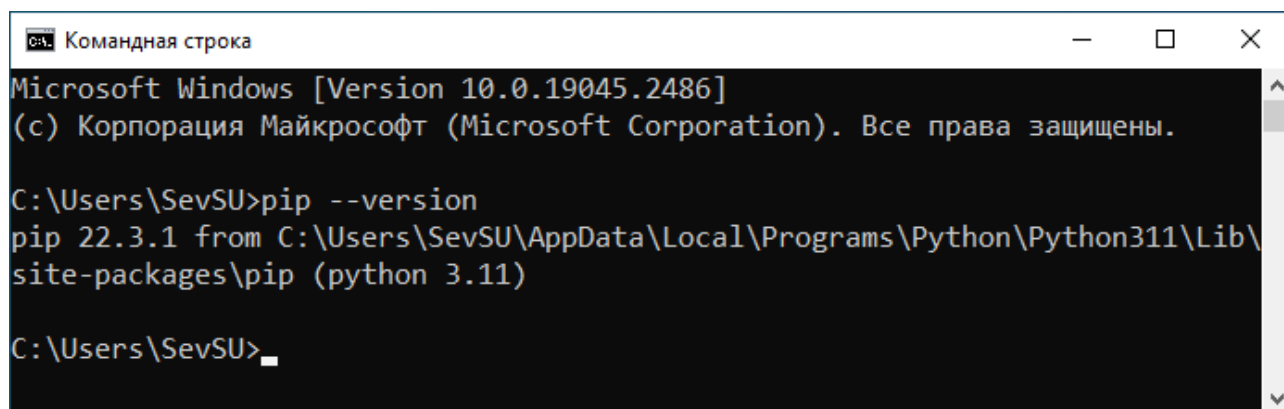
№	Команда	Назначение	Пример
1	install	Установка библиотеки	<code>pip install libname</code>
2	install --upgrade	Обновление библиотеки	<code>pip install --upgrade libname</code>
3	uninstall	Удаление библиотеки	<code>pip uninstall libname</code>

Если требуется выполнить установку нескольких пакетов одновременно, то можно воспользоваться следующим приёмом. Создаётся файл `requirements.txt` в котором все необходимые для установки библиотеки перечисляются в каждой строке файла. После создания файла, для установки пакета библиотек необходимо выполнить команду

`pip install -r requirements.txt`

Для того чтобы правильно указать формат команды или имя библиотеки, рекомендуется обратиться к ресурсу <https://pypi.org/>

Пример. Выполним установки библиотеки `xgboost`. Перед установкой библиотеки убедимся в том, что менеджер пакетов *pip* установлен в системе, так как это показано на рис. 40.



```

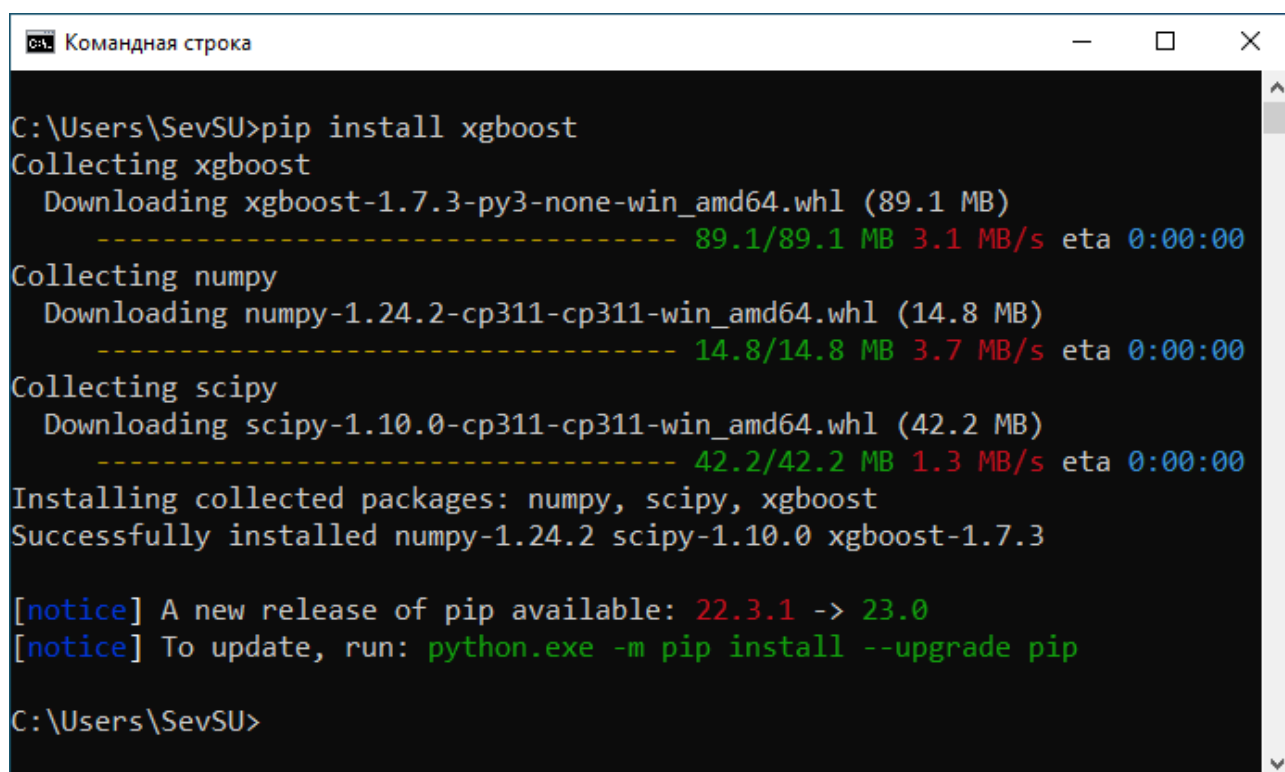
Командная строка
Microsoft Windows [Version 10.0.19045.2486]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\SevSU>pip --version
pip 22.3.1 from C:\Users\SevSU\AppData\Local\Programs\Python\Python311\Lib\
site-packages\pip (python 3.11)

C:\Users\SevSU>_
  
```

Рис. 40 — Проверка установки менеджера пакетов *pip*

Как видно из результатов работы команды, в системе установлен менеджер пакетов версии 22.3.1, следовательно, можно выполнить установку библиотеки. Результат выполнения команды установки приведён на рис. 41.



```
C:\Users\SevSU>pip install xgboost
Collecting xgboost
  Downloading xgboost-1.7.3-py3-none-win_amd64.whl (89.1 MB)
----- 89.1/89.1 MB 3.1 MB/s eta 0:00:00
Collecting numpy
  Downloading numpy-1.24.2-cp311-cp311-win_amd64.whl (14.8 MB)
----- 14.8/14.8 MB 3.7 MB/s eta 0:00:00
Collecting scipy
  Downloading scipy-1.10.0-cp311-cp311-win_amd64.whl (42.2 MB)
----- 42.2/42.2 MB 1.3 MB/s eta 0:00:00
Installing collected packages: numpy, scipy, xgboost
Successfully installed numpy-1.24.2 scipy-1.10.0 xgboost-1.7.3

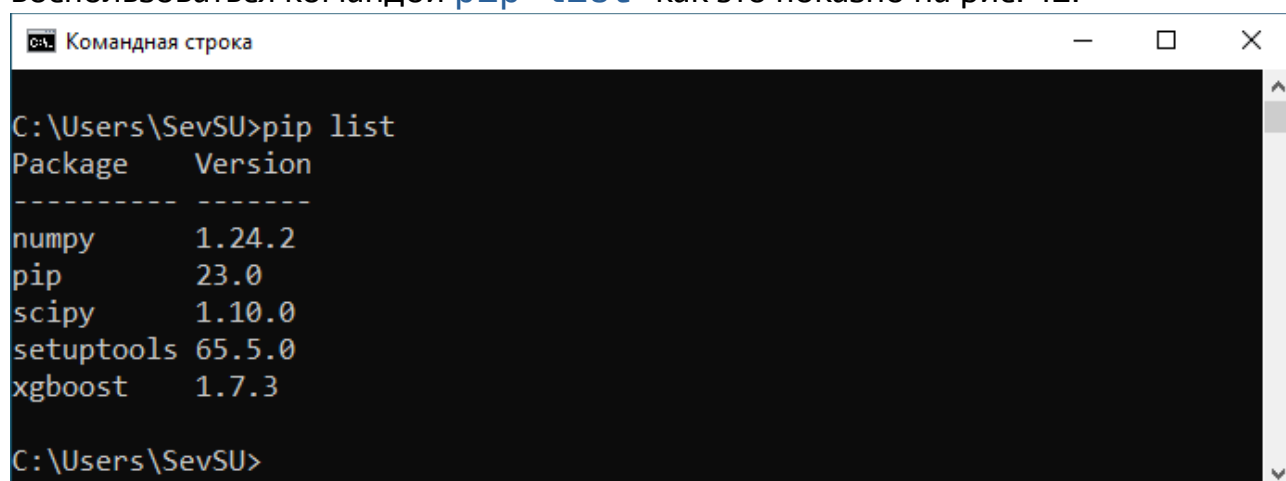
[notice] A new release of pip available: 22.3.1 -> 23.0
[notice] To update, run: python.exe -m pip install --upgrade pip

C:\Users\SevSU>
```

Рис. 41 — Установка библиотеки xgboost

На рис. 41 видно, что при установке библиотеки xgboost были установлены зависимости — библиотеки, которые необходимы для нормальной работы устанавливаемой библиотеки. Также, пользователю предлагается обновиться до актуальной версии менеджера пакетов.

Для просмотра всех библиотек, установленных в системе, можно воспользоваться командой `pip list` как это показано на рис. 42.



```
C:\Users\SevSU>pip list
Package      Version
-----
numpy        1.24.2
pip          23.0
scipy        1.10.0
setuptools   65.5.0
xgboost      1.7.3

C:\Users\SevSU>
```

Рис. 42 — Получение списка установленных библиотек

Помимо перечня библиотек, пользователю доступна информация о версии установленной библиотеки. При необходимости, библиотека может быть обновлена с помощью команды `upgrade`.

Основы Python

Python считается скриптовым языком. Но это не означает, что на Python можно писать лишь небольшие фрагменты программ. Python – это полноценный язык программирования, поддерживающий модули, библиотеки, объектно-ориентированный подход и многое другое. В тоже время нужно помнить что Python – интерпретируемый язык (т. е. язык программирования, исходный код которого выполняется программой-интерпретатором, скомпилировать код с такого языка в исполняемый файл нельзя).

Язык Python отличается удобочитаемостью и простотой. Для структурирования кода, в отличие от большинства других языков программирования, используются пробелы, а не фигурные скобки. Двоеточием обозначается начало блока кода с отступом, весь последующий код до конца блока должен быть набран с точно таким же отступом. Например, вот как выглядит форматирование кода с помощью пробелов для реализации алгоритма быстрой сортировки:

```
for x in array:
    ....if x < pivot:
    .....less.append(x)
    ....else:
    .....greater.append(x)
```

Как было сказано выше, Python более лаконичен. Сравним для примера код на языке C++ и код на Python, который выводит на печать строки из предопределенного списка.

Пример: Код на C++

```

1  #include <iostream>
2
3  const std::string scientists[] = {
4      "Isaac Newton",
5      "Mikhail Lomonosov",
6      "Marie Curie",
7      "Albert Einstein" };
8
9  int main()
10 {
11     for (int i = 0; i < scientists->length(); i++)
12     {
13         std::cout << scientists[i] << std::endl;
14     }
15
16     return EXIT_SUCCESS;
17 }

```

Пример: Код на Python

```

In [1]: scientists = [
        "Isaac Newton",
        "Mikhail Lomonosov",
        "Marie Curie",
        "Albert Einstein"]
print(scientists)

['Isaac Newton', 'Mikhail Lomonosov', 'Marie Curie', 'Albert Einstein']

```

или с использованием цикла for

```

In [2]: scientists = [
        "Isaac Newton",
        "Mikhail Lomonosov",
        "Marie Curie",
        "Albert Einstein"]
for person in scientists:
    print(person)

Isaac Newton
Mikhail Lomonosov
Marie Curie
Albert Einstein

```

В приведенном примере код на python не требует определения главной функции, не нуждается в дополнительных библиотеках, не использует дополнительное кодовое пространство для идентификации блоков кода.

Преимущества Python:

- простой и понятный синтаксис;
- поддержка кросс-платформенной разработки (Windows, Unix, Linux, macOS);

- бесплатное использование (в отличие от коммерческих систем научных расчетов Matlab, Mathcad, Mathematica);
- большое количество дополнительных специализированных библиотек (которые тоже бесплатны);
- язык прост в изучении (в отличие от C++);
- язык очень гибок и поддерживает различные парадигмы (процедурное программирование, объектно-ориентированное, функциональное).

Недостатки Python:


- скорость выполнения программ на Python несколько ниже чем на C++ (однако есть библиотеки Python, которые хорошо оптимизированы по скорости, например NumPy);
- исходные коды алгоритмов, реализованных на Python невозможно скрыть (потенциальные проблемы, связанные с авторскими правами на результаты интеллектуальной деятельности);
- проблемы совместимости со старыми стандартами языка (Python 2 vs Python 3).

Типы данных в Python

Важная характеристика языка Python — последовательность его объектной модели. Все числа, строки, структуры данных, функции, классы, модули и т. д. в интерпретаторе заключены в «ящики», которые называются объектами Python. С каждым объектом ассоциирован тип (например, строка, число или функция) и внутренние данные. Это делает язык более гибким.

Различают числовые типы данных, к которым в Python относятся целые числа (type: int), вещественные числа (type: float) и комплексные числа (type: complex). При этом, во время определения переменной тип можно не задавать. Более того, одна и та же переменная может быть использована для хранения различных типов данных.

Пример:

```
In [3]:  # Целое число
        a = 5
        type(a)

Out[3]: int
```

```
In [4]: # Вещественное число  
a = 5.25  
type(a)
```

Out[4]: float

```
In [5]: # Комплексное число  
a = 5+0.25j  
type(a)
```

Out[5]: complex

По аналогии с другими языками программирования допускается приведение типов.

Пример:

```
In [7]: # Приведение к целому  
a = int(5.25)  
print(a)
```

5

```
In [8]: # Приведение к вещественному  
a = float(5.0)  
print(a)
```

5.0

```
In [9]: # Приведение к комплексному  
a = complex(5.25)  
print(a)
```

(5.25+0j)

Основные арифметические операции

В Python используются следующие арифметические операции

- + Сложение
- Вычитание
- * Умножение
- / Вещественное деление
- // Целочисленное деление
- % Остаток от деления
- ** Возведение в степень

При этом наивысшим приоритетом обладает операция возведения в степень **, затем идут операции * / // %, наименьший приоритет у операций + -.

Числа в Python это тоже объекты. Любой объект представляет собой данные, а также набор атрибутов и методов для работы с данными. В общем виде это можно представить в виде записи `<object>.<attribute>` и `<object>.<method>()`. Например комплексные числа имеют встроенные атрибуты для доступа к вещественной и мнимой части числа.

```
In [10]:  ► a = (5+0.25j).real  
          print(a)  
          5.0
```

```
In [11]:  ► b = (5+0.25j).imag  
          print(b)  
          0.25
```

В качестве демонстрации работы метода рассмотрим метод получения комплексно-сопряженного числа.

```
In [12]:  ► (5+0.25j).conjugate()  
Out[12]: (5-0.25j)
```

Для работы с числовыми данными в Python есть ряд встроенных функций (например *round*, *abs*) но удобнее пользоваться специализированными библиотеками, например *math*, описание которой приведено в следующем разделе данного пособия.

Переменные в Python

Существует несколько правил для назначения имен переменных в Python:

- имена переменных зависят от регистра (переменные *parameter_vauie* и *Parameter_Value* – разные переменные);
- имена переменных могут содержать буквы, символы подчеркивания «_» и цифры 0..9;
- имена переменных не могут начинаться с цифры;

— в качестве имен переменных не могут использоваться зарезервированные слова (см. таблицу ниже).

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	False	True	None	

Помимо обязательных правил, существует набор правил — рекомендаций для выбора имен переменных:

- имя переменной должно быть осмысленным (для обозначения площади лучше использовать имя *area* вместо сокращения *a*);
- имя переменной не должно быть слишком длинным (например *the_area_of_the_triangle* в этом случае лучше сократить до *triangle_area*);
- не рекомендуется использовать в качестве имени переменной символы *I* и *O* в верхнем регистре, и символ *l* в нижнем регистре, поскольку их легко спутать с 1 и 0 соответственно;
- имена переменных *i*, *j*, *k* обычно используются только в циклах или для обозначения индекса;
- в Python не рекомендуется использовать верблюжью нотацию вместо этого рекомендуется использовать имена переменных в нижнем регистре, разделенные нижним подчеркиванием (*mean_value* вместо *meanValue*).

С полным руководством по стилям форматирования кода Python можно ознакомиться по ссылке <https://peps.python.org/pep-0008/> (PEP 8 – Style Guide for Python Code).

Пример: Вычисление площади треугольника по формуле Герона

$$S = \sqrt{p(p-a)(p-b)(p-c)}; p = \frac{a+b+c}{2}.$$

Обратите внимание, что перед использованием функций модуля *math* функции этого модуля должны быть импортированы в скрипт с помощью инструкции.

```
import math
```



```
In [12]: # Вычисление площади треугольника по формуле Герона
import math

side_a = 4.5
side_b = 2.3
side_c = 3.9

semi_perimeter = (side_a + side_b + side_c) / 2
triangle_area = math.sqrt(semi_perimeter
                           * (semi_perimeter - side_a)
                           * (semi_perimeter - side_b)
                           * (semi_perimeter - side_c))

print(f"Площадь треугольника: {triangle_area:.2f}")

Площадь треугольника: 4.48
```

Логический тип данных и операторы сравнения

Логический тип данных в Python (type: bool) может принимать два значения *True* и *False* (обратите внимание, что значение логических переменных задается с большой буквы).

```
In [13]: # Определение логической переменной
bool_value = True
type(bool_value)
```

Out[13]: bool

Для выполнения операций сравнения используются следующие операторы

==	Равно
!=	Не равно
>	Больше чем
<	Меньше чем
>=	Больше или равно
<=	Меньше или равно

Результатом выполнения логической операции является также логический объект, принимающий значение *True* или *False*.

При выполнении операции *Равно* для вещественных чисел, нужно проявлять осторожность, поскольку сами значения хранятся не точно или с потерей точности, например

```
In [14]: numberA = 0.01  
         numberB = 0.1**2  
         numberA == numberB
```

Out[14]: False

Для первой переменной возможно что значение переменной *numberA* равно 0.010000000000000000208 в то время как значение переменной *numberB* равно 0.010000000000000000194 поэтому результат сравнения *False*.

Начиная с Python 3.5 библиотека *math* предоставляет в распоряжение разработчика функцию *isclose*, которую можно использовать для решения указанной выше проблемы.

Пример:

```
In [15]: math.isclose(numberA, numberB)
```

Out[15]: True

```
In [16]: math.isclose(numberA, numberB, abs_tol = 0.001)
```

Out[16]: True

Используя атрибут функции *abs_tol* можно задать точность, которая будет использоваться для сравнения вещественных чисел (см. ячейку 16 на рисунке выше).

Для комбинации различных логических условий в языке Python используются ключевые слова *and*, *or* и *not*. При этом операнды не обязательно помещать в круглые скобки.

Пример:

```
In [17]: 1.25 > 2.43 or 1.25 > -0.37
```

Out[17]: True

Условный оператор if-elif-else

Условный оператор в Python в общем случае можно представить в виде:

```
if <if-condition>:  
    ....<some code>  
elif <elif-condition>:  
    ....<some code>  
else <else-condition>:  
    ....<some code>
```

В отличие от синтаксиса языка C++ в Python отсутствует оператор *switch*. Поэтому условный оператор *if* в Python несколько отличается от классической конструкции. Помимо известных *if* и *else* здесь присутствует еще одна конструкция — *elif*. Причем блоков кода *elif* может быть несколько, тогда проблема отсутствия оператора *switch* решается достаточно просто.

Пример:

```
In [1]:  number = int(input('Введите целое число от 1 до 3: '))  
        if number == 1:  
            print("Вы ввели единицу")  
        elif number == 2:  
            print("Вы ввели двойку")  
        elif number == 3:  
            print("Вы ввели тройку")  
        else:  
            print("Ошибка ввода данных!")
```

```
Введите целое число от 1 до 3: 2  
Вы ввели двойку
```


В приведенном фрагменте кода используется конструкция `input`, она может быть использована для случаев, когда требуется организовать интерактивный ввод данных от пользователя в блокноте Jupyter Notebook.

Оператор цикла `while`

Оператор `while` позволяет многократно выполнять тело цикла до тех пор, пока выполняется условие, указанное после ключевого слова `while`. Также оператор может иметь необязательную конструкцию `else`.

```
while <condition>:  
....<some code>  
else <else-condition>:  
....<some code>
```

Пример:

```
In [19]:  # Подключаем библиотеку  
# для генерации случайных чисел  
import random  
number = random.randint(0, 9) # Генерация числа  
counter = 1; # Счетчик попыток  
while int(input('Ваш вариант: ')) != number:  
    counter = counter + 1  
else:  
    print(f'Вы угадали число {number} за {counter} попыток')
```

```
Ваш вариант: 7  
Ваш вариант: 1  
Ваш вариант: 2  
Ваш вариант: 3  
Ваш вариант: 4  
Ваш вариант: 5  
Ваш вариант: 6  
Вы угадали число 6 за 7 попыток
```

В приведенном фрагменте кода вначале, с помощью библиотеки `random`, генерируется случайное число в диапазоне от 0 до 9, после чего пользователю предлагается ввести свой вариант. Если пользователь не угадал число, в теле цикла происходит инкремент счетчика попыток. После того как пользователь ввел правильное значение, работа цикла завершается и выводится соответствующее информационное сообщение.

Оператор цикла for

Как известно из курса программирования на языке C++ оператор цикла *for* используется в тех случаях, когда число итераций тела цикла заранее известно. Но синтаксис оператора несколько отличается и имеет вид:

```
for <var> in <iterable>:  
....<some code>
```

Также как и в C++ работу тела цикла можно прервать с помощью оператора *break*.

Пример:

```
In [20]: ► fruits = ["apple", "banana", "cherry", "potato"]  
         for f in fruits:  
             if f != "potato":  
                 print(f)  
             else:  
                 break  
  
apple  
banana  
cherry
```

В цикле перебираются и выводятся на экран названия фруктов, до тех пор, пока не встретится название potato.

Если нужно организовать цикл, например от 1 до 3, то соответствующая конструкция будет иметь вид:

```
In [21]: ► n = 3  
         for i in range(1, n + 1):  
             print(i)  
  
1  
2  
3
```

Здесь отдельного внимания занимает функция *range*. Эта функция используется для генерации множества значений, которое начинается с числа, равного первому аргументу функции и заканчивается числом на единицу меньше второго аргумента функции.

В общем виде функция имеет вид:

`range(start, stop, step)`

где *start* (необязательный параметр) целое число, определяющее начало последовательности (по умолчанию равен 0);

stop (обязательный параметр) целое число, определяющее конец диапазона (не входит в диапазон);

step (необязательный параметр) целое число, определяющее шаг изменения последовательности (по умолчанию равен 1).

Исходя из сказанного, вызов функции `range(3)` приведет к генерации последовательности `{0, 1, 2}`.

Важное замечание: функцию `range` нельзя использовать для генерации последовательности с дробным шагом.

Пользовательские функции

Для определения пользовательской функции в языке Python предусмотрено ключевое слово `def`. Далее указывается имя функции и при необходимости перечень входных параметров. В отличие от C++ указывать тип данных входных параметров не нужно. В конце определения ставится точка с запятой.

Рассмотрим синтаксис создания и вызова пользовательской функции, возвращающей квадрат входного аргумента.

Пример:

```
In [22]: ► def funcX2(x):  
          return x**2  
  
          print(funcX2(5))  
          print(funcX2(0.25))  
  
          25  
          0.0625
```

В данном примере мы определили пользовательскую функцию `funcX2`, которая принимает на вход один параметр `x`, тип которого не указывается. Затем, функция вызывается для целочисленного значения параметра и вещественного значения параметра. Однако вызов этой же функции для строкового значения параметра `print(funcX2('Mup'))` приведет к ошибке, поскольку операция `**` не может быть применена к строкам.

Библиотека math

В языке Python существует ряд стандартных библиотечных модулей, которые могут быть использованы для расширения базовых функций языка. К таким библиотечным модулям относится модуль `math` который экспортирует стандартные средства из библиотеки математических функций C для применения в Python. В табл. 2 и табл. 3 перечислены математические функции и константы, экспортируемые этим модулем. Для получения детального описания той или иной функции библиотеки необходимо импортировать модуль `math` и вызвать функцию `help(math.имя_функции)`. В результате будут выведено описание функции, сведения об аргументах и примечания к её применению. Подробнее о модуле `math` см. в руководстве по библиотеке Python <https://docs.python.org/3/library/math.html>

Табл. 2 — Математические функции модуля `math`

Функция	Описание
<code>math.acos(x)</code>	Возвращает арккосинус числа x
<code>math.acosh(x)</code>	Возвращает обратный гиперболический косинус числа x
<code>math.asin(x)</code>	Возвращает арксинус числа x
<code>math.asinh(x)</code>	Возвращает обратный гиперболический синус числа x
<code>math.atan(x)</code>	Возвращает арктангенс числа в радианах x
<code>math.atan2(y, x)</code>	Возвращает арктангенс y / x в радианах
<code>math.atanh(x)</code>	Возвращает обратный гиперболический тангенс числа x
<code>math.ceil(x)</code>	Округляет число x до ближайшего целого в большую сторону
<code>math.comb(n, k)</code>	Возвращает общее число возможных комбинаций из n по k
<code>math.copysign(x, y)</code>	Возвращает вещественное число, состоящее из значения первого параметра x и знака второго параметра y
<code>math.cos(x)</code>	Возвращает косинус числа x
<code>math.cosh(x)</code>	Возвращает гиперболический косинус числа x
<code>math.degrees(x)</code>	Переводит значение угла x из радиан в градусы
<code>math.dist(p, q)</code>	Возвращает Евклидово расстояние между двумя точками p и q , где p и q — координаты точек
<code>math.erf(x)</code>	Вычисляет функцию ошибок Гаусса числа x
<code>math.erfc(x)</code>	Возвращает дополнительную функцию ошибки числа x
<code>math.exp(x)</code>	Возвращает значение e^x
<code>math.expm1(x)</code>	Возвращает значение $e^x - 1$
<code>math.fabs(x)</code>	Возвращает абсолютное значение числа x
<code>math.factorial(x)</code>	Возвращает факториал числа x

Табл. 2 (продолжение) — Математические функции модуля *math*

Функция	Описание
<code>math.floor(x)</code>	Округляет число x до ближайшего целого в меньшую сторону
<code>math.fmod(x, y)</code>	Возвращает остаток от деления x / y
<code>math.frexp(x)</code>	Возвращает мантису и экспоненту числа x ($x = m \cdot (2^{**}e)$)
<code>math.fsum(iterable)</code>	Возвращает сумму всех элементов в последовательности <code>iterable</code>
<code>math.gamma(x)</code>	Возвращает гамма функцию числа x
<code>math.gcd(int1, int2)</code>	Возвращает наибольший общий делитель двух целых чисел <code>int1</code> и <code>int2</code>
<code>math.hypot(x1, x2, x3, ..., xN)</code>	Возвращает Евклидову норму
<code>math.isclose(a, b)</code>	Проверяет, близки ли друг другу два числа a и b
<code>math.isfinite(x)</code>	Проверяет, является x ли число конечным
<code>math.isinf(x)</code>	Проверяет, является x ли число бесконечным
<code>math.isnan(x)</code>	Проверяет, является x ли число NaN
<code>math.isqrt(x)</code>	Округляет квадратный корень числа x до ближайшего наименьшего целого
<code>math.ldexp(x, i)</code>	Возвращает обратное значение функции <code>math.frexp()</code> равное $x * (2^{**}i)$
<code>math.lgamma(x)</code>	Возвращает натуральный логарифм гамма значения числа x
<code>math.log(x, base)</code>	Возвращает натуральный логарифм числа x или логарифм по основанию <code>base</code>
<code>math.log10(x)</code>	Возвращает логарифм по основанию 10 для числа
<code>math.log1p(x)</code>	Возвращает натуральный логарифм числа $x+1$
<code>math.log2(x)</code>	Возвращает логарифм по основанию 2 для числа
<code>math.perm(n, k)</code>	Возвращает число сочетаний из n по k
<code>math.pow(x, y)</code>	Возвращает x в степени y
<code>math.prod(iterable, start)</code>	Возвращает произведение чисел в последовательности <code>iterable</code> , начиная с элемента с номером <code>start</code>
<code>math.radians(x)</code>	Переводит значение угла из градуса в радианы
<code>math.remainder(x, y)</code>	Возвращает остаток от деления числа x на y
<code>math.sin(x)</code>	Возвращает синус числа x
<code>math.sinh(x)</code>	Возвращает гиперболический синус числа x
<code>math.sqrt(x)</code>	Возвращает квадратный корень числа x
<code>math.tan(x)</code>	Возвращает тангенс числа x
<code>math.tanh(x)</code>	Возвращает гиперболический тангенс числа x
<code>math.trunc(x)</code>	Возвращает целую часть числа x

Табл. 3 — Математические константы модуля math

Константа	Описание
math.e	Возвращает число Эйлера (2.7182...)
math.inf	Возвращает вещественное число, эквивалентное Inf+
math.nan	Возвращает вещественное число NaN
math.pi	Возвращает число PI (3.1415...)
math.tau	Возвращает число tau (6.2831...)

Работа с файлами

При работе с фалами чаще всего встречаются две задачи: считывание информации из файла и запись информации в файл. В Python эти задачи решаются достаточно просто за счёт использования специальных файловых объектов. Независимо от типа операции с фалом, вначале необходимо создать файловый объект, после чего файл нужно открыть. Для этого существует специальная функция `open()`. Первым обязательным аргументом функции является полное имя файла, с которым планируется работа, следующим необязательным параметром является ключ, определяющий спецификатор доступа. Возможные значения ключа приведены в табл. 4. Созданный в результате вызова функции `open` файловый объект имеет специальные методы для выполнения основных действий — чтения `read()` и записи `write()`. После того, как работа с файловым объектом завершена, необходимо вызвать метод `close()`.

Табл. 4 — Значения ключей для доступа к файлу

Ключ	Значение
r	Режим чтения содержимого файла. Используется по умолчанию, если режим доступа явно не указан
w	Режим записи в файл. При использовании ключа w содержимое файла уничтожается (если файл уже был создан ранее)
x	Режим создания файла с возможностью записи данных в файл. Если файл уже существует, возникает ошибка
a	Режим записи в файл. Если файл существует, то новое содержимое дописывается в конец файла
b	Режим доступа к бинарному файлу
t	Режим доступа к текстовому файлу. Используется по умолчанию, если явно режим доступа не задан

Варианты задания

В соответствии с вариантом задания вычислить значение математической функции. Начальное и конечное значения аргумента, а также шаг изменения аргумента использовать в соответствии со значениями, представленными в табл. 5. Для каждой функции определить максимальное и минимальное значения. Результат табулирования функции вывести в текстовый файл, при выводе вещественных значений ограничиться двумя знаками после запятой для значения аргумента и тремя знаками для значения функции.

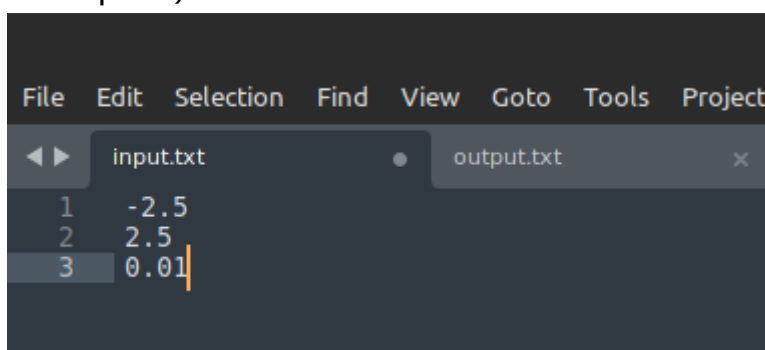
Табл. 5 — Варианты задания

№	Функция	x_0	x_f	h_x
1	$y(x) = 3x^3 - 15x^2 - 12x + 8$	-5	15	0,01
2	$y(x) = x^2 \sin^3(3x)$	-3	2,5	0,001
3	$y(x) = x^3 \sin^5(4x)$	-1,6	3,2	0,001
4	$y(x) = (1 + 6 \sin^2(x))^{1/2} \cos(3x)$	-5	12	0,001
5	$y(x) = 2x^4 - 12x^3 - 4x^2 + 2$	-5	10	0,01
6	$y(x) = \cos(2,5x) \sin(3x) - 0,2$	-7	7	0,01
7	$y(x) = 4^{-(x-1)^2} \cos(2x)$	-10	20	0,1
8	$y(x) = \cos(2,5x) \sin(3x) - 0,2$	-3,2	8	0,001
9	$y(x) = x(\sin(3x) + \cos(2x))$	-3,5	4,5	0,01
10	$y(x) = x^2(\sin^2(3x) - \cos^2(2x))$	-3,5	5,5	0,1

Пример выполнения работы

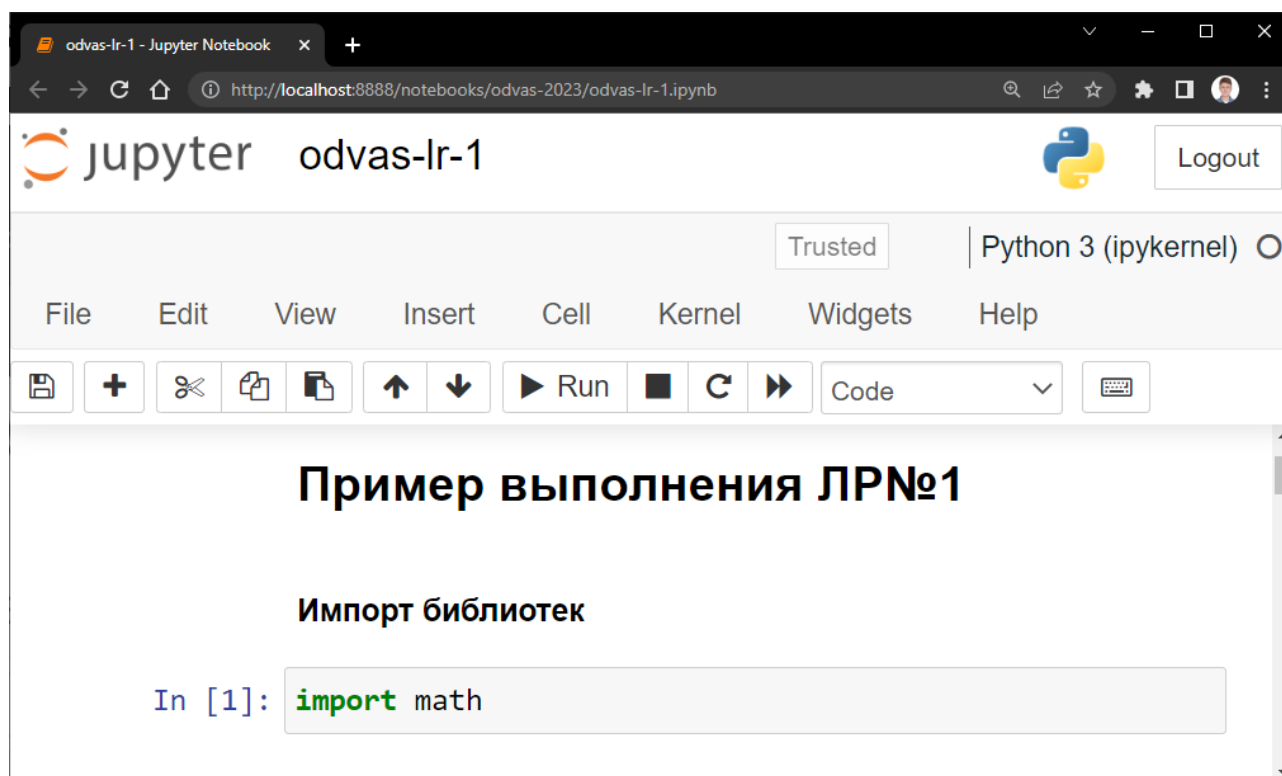
Вычислить значение функции $y(x)=x^3 \sin(|4x|)$ на интервале от $x_0=-2,5$ до $x_f=2,5$ с шагом $h_x=0,01$. Значения x_0, x_f, h_x прочитать из входного файла. Результат табулирования функции вывести в текстовый файл, при выводе вещественных значений ограничиться двум знаками после запятой для значения аргумента и тремя знаками для значения функции. Математическую функцию определить с помощью блока **def**.

Входной файл данных имеет вид (каждое новое значение параметра начинается с новой строки)



```
File Edit Selection Find View Goto Tools Project
input.txt output.txt x
1 -2.5
2 2.5
3 0.01
```

Реализация задания выполнена в Jupyter Notebook. Ячейка [1] предназначена для импорта библиотеки `math`.



Далее осуществляется чтение исходных данных из файла

Чтение исходных данных

```
In [2]: in_file_name = 'input.txt' # Имя входного файла
in_file = open(in_file_name, 'r') # Открытие файла
X0 = in_file.readline().strip() # Читаем X0
Xf = in_file.readline().strip() # Читаем Xf
hX = in_file.readline().strip() # Читаем hX
# Выводим считанные значения параметров
print(f'X0 = {X0}; Xf = {Xf}; hX = {hX}')
in_file.close() # Закрываем файл
```

```
X0 = -2.5; Xf = 2.5; hX = 0.01
```

Поскольку данные хранятся в текстовом виде, при чтении из файла необходимо удалить служебные символы, такие как возврат каретки (перевод на новую строку). Для этого можно воспользоваться функцией `strip()`. Для вывода результата в консоль рекомендуется использовать строки с форматированием, которые сочетают в себе статический и динамический текст. Переменные, которые должны быть выведены в строке помещаются в фигурные скобки.

Следующее действие — необходимо определить пользовательскую функцию. Для этого используется служебное слово **def**.

Определяем пользовательскую функцию

```
In [3]: def my_func(X):
        return math.pow(X,3)*math.sin(math.fabs(4*X))
```

Далее происходит преобразование входных параметров из текстового формата в числовой. При этом могут быть использованы те же самые переменные. Также осуществляется расчёт общего количество точек, для которых должно быть вычислено значение функции.

Табулирование функции и запись в файл

```
In [4]: # Преобразование в числовой формат
Xf = float(Xf)
X0 = float(X0)
hX = float(hX)
N = math.ceil((Xf - X0)/hX) + 1
print(f'Кол-во точек: {N}')
```

```
Кол-во точек: 501
```

В ячейке [5] происходит основная процедура табулирования функции. При этом используется функция `range()` для генерации набора целых чисел в диапазоне от 0 до 500. В результате чего по заданному значению изменения шага аргумента и текущему номеру отсчёта i осуществляется вычисление текущего значения аргумента. Инициализация переменных для поиска минимального и максимального значения функции производится с помощью библиотеки `math`. Для вывода в файл используется форматирование строки и ограничение количества знаков после запятой. Это обеспечивается за счёт применения конструкции `{X:.2f}` которая означает вывод вещественного числа с двумя знаками после запятой. Для вывода значения функции используется конструкция `{Y:.4f}` что соответствует четырём знакам после запятой. Поиск максимума и минимума осуществляется с помощью оператора `if`.

```
In [5]: out_file_name = 'output.txt' # Имя входного файла
out_file = open(out_file_name, 'w') # Открытие файла
# Инициализируем переменные для максимума и минимума
Ymin = math.inf
Ymax = -math.inf
for i in range(N):
    X = X0 + i*hX # Вычисление аргумента
    Y = my_func(X) # Вычисление функции
    # Вывод в файл
    out_file.write(f'{X:.2f}\t{Y:.4f}\n')
    # Поиск минимума и максимума
    if (Y < Ymin):
        Ymin = Y

    if (Y > Ymax):
        Ymax = Y

out_file.close()
```

```
In [6]: print(f'Минимальное значение: {Ymin:.4f}')
print(f'Максимальное значение: {Ymax:.4f}')
```

Минимальное значение: -8.5003

Максимальное значение: 8.5003

В результате работы программы, формируется выходной файл `output.txt` который имеет вид

1	-2.50	8.5003
2	-2.49	7.8740
3	-2.48	7.2486
4	-2.47	6.6254
5	-2.46	6.0053

Контрольные вопросы

1. Назовите современные программные инструментальные средства для анализа данных. Дайте их краткую характеристику.
2. Что из себя представляет блокнот Jupyter Notebook?
3. Сравните возможности Jupyter Notebook и PyCharm. Какой из этих инструментов вы выбираете и почему?
4. Перечислите основные команды / горячие клавиши Jupyter Notebook.
5. Что такое пакеты?
6. Как узнать, какие пакеты установлены?
7. Как установить необходимые пакеты?
8. Что такое зависимость пакетов?
9. Что такое общесистемное и виртуальное окружение проекта Python?
10. Как использовать Jupyter Notebook для написания скрипта на языке Python?
11. Как создаётся проект Python в IDE PyCharm?
12. Как осуществляется вывод в файл с помощью Python?
13. Назовите основные правила по написанию кода на Python, перечислите основные синтаксические конструкции.

Список использованных источников

1. **Маккини У. Python и анализ данных / У. Маккини** — Пер. с англ. — М.: ДМК Прес, 2020. — 540 с.
2. **Плас Дж. В. Python для сложных задач: наука о данных и машинное обучение / Дж. В. Плас** — Пер. с англ. — СПб.: Питер, 2022. — 576 с.
3. **Грас Дж. Data Science. Наука о данных с нуля / Дж. Грас** — Пер. с англ. — 2-е издание — СПб.: БХВ-Петербург, 2022. — 416 с.
4. **Луц М. Изучаем Python. Том I / М. Луц** — Пер. с англ. — 5-е издание — СПб.: ООО «Диалектика», 2020. — 720 с.
5. **Луц М. Изучаем Python. Том II / М. Луц** — Пер. с англ. — 5-е издание — СПб.: ООО «Диалектика», 2020. — 832 с.
6. **Рашка С. Python и машинное обучение: машинное обучение с использованием Python, scikit-learn и TensorFlow / С. Рашка, В. Мирджалили** — Пер. с англ. — 3-е издание — СПб.: ООО «Диалектика», 2020. — 848 с.
7. **Брюс П. Практическая статистика для специалистов Data Science / П. Брюс, Э. Брюс, П. Гадек** — Пер. с англ. — 2-е издание — СПб.: БХВ-Петербург, 2022. — 352 с.
8. **Васильев А. Программирование на Python в примерах и задачах / А. Васильев** — М.: Эксмо, 2021. — 616 с.
9. **Луц М. Python. Карманный справочник / М. Луц** — Пер. с англ. — 5-е издание — СПб.: ООО «Диалектика», 2020. — 320 с.

Полезные ссылки

1. <https://linuxmint.com/> — Страница загрузки дистрибутива Linux Mint.
2. <https://ubuntu.com/desktop> — Страница загрузки дистрибутива Ubuntu Desktop.
3. <https://www.python.org/> — Страница загрузки установщика интерпретатора Python.
4. <https://www.jetbrains.com/pycharm/> — Страница загрузки установщика IDE PyCharm Community Edition.
5. <https://www.anaconda.com/> — Страница загрузки установщика среды Anaconda.
6. <https://pypi.org/> — Каталог библиотек Python (Python Package Index).
7. <https://docs.python.org/3/library/functions.html> — Документация по функциям Python.
8. <https://docs.python.org/3/library/math.html> — Документация по библиотеке math.

Учебное издание

Василий Викторович Альчаков

**ПРОГРАММНЫЕ СРЕДСТВА ОБРАБОТКИ И АНАЛИЗА ДАННЫХ НА PYTHON.
ОСНОВЫ PYTHON**

Методические указания к выполнению лабораторной работы

Оригинал-макет и вёрстка В.В. Альчаков

© СевГУ, 2024

© Альчаков В.В., 2024