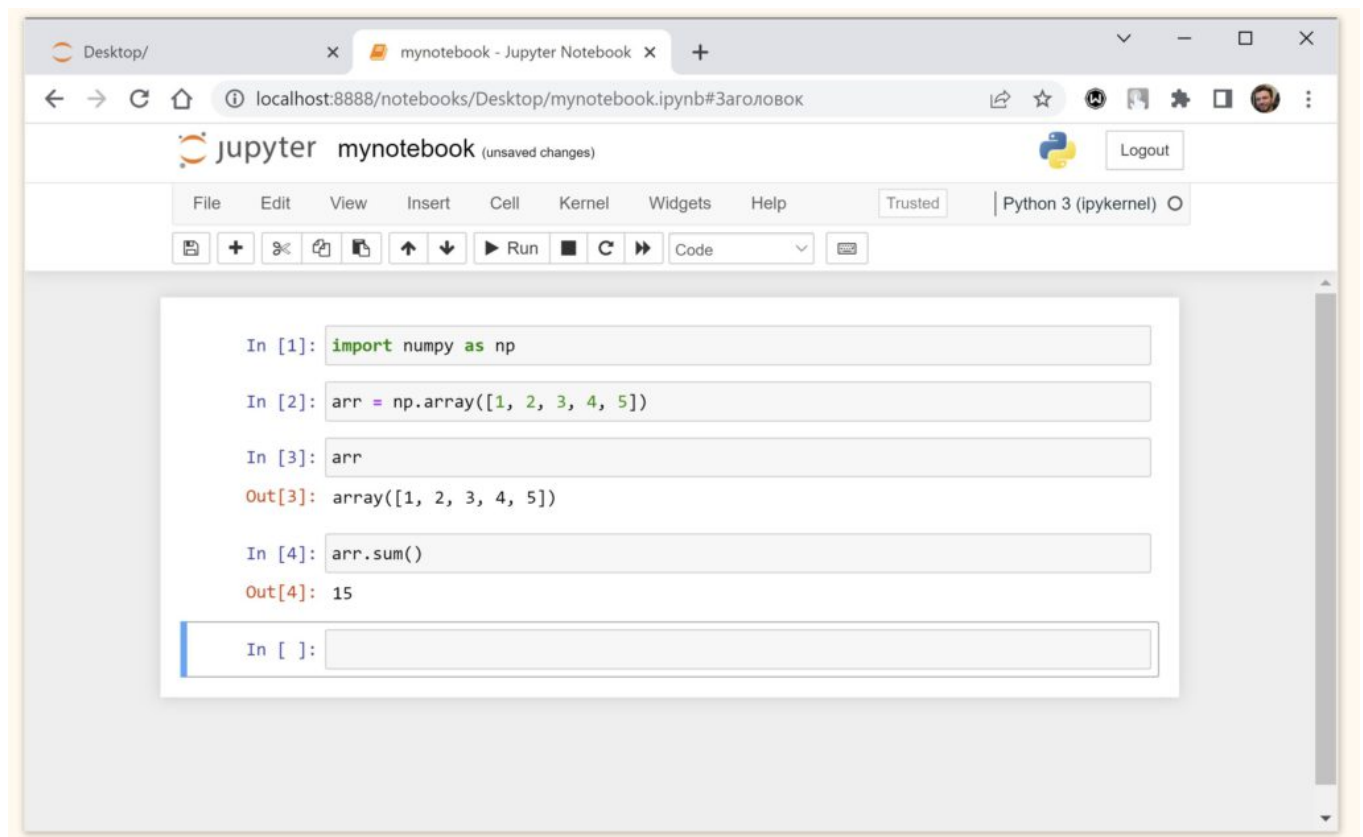


Jupyter Notebook

[Все курсы](#) > [Программирование на Питоне](#) > [Занятие 14](#)



Программа Jupyter Notebook — это локальная программа, которая открывается в браузере и позволяет интерактивно исполнять код на Питоне, записанный в последовательности ячеек.



Облачной версией Jupyter Notebook является программа Google Colab, которой мы уже давно пользуемся на курсах машинного обучения. Если вы проходили мои занятия, то в работе с этой программой для вас не будет почти ничего нового.

Содержание занятия

- Как установить Jupyter Notebook
 - Anaconda
 - Установка дистрибутива Anaconda на Windows
 - Как запустить Jupyter Notebook
- Особенности работы
 - Код на Python
 - Установка новых пакетов
 - Два Питона на одном компьютере
 - Markdown в Jupyter Notebook
 - Формулы на LaTeX
 - Программирование на R
- Подробнее про Anaconda
 - Conda
 - Anaconda Prompt
 - Anaconda Navigator
 - JupyterLab
- Подведем итог
 - Вопросы для закрепления
 - Ответы на вопросы

Как установить Jupyter Notebook

Способ 1. Если на вашем компьютере уже установлен Питон, то установить Jupyter Notebook можно через менеджер пакетов `pip`.

Способ 2 (рекомендуется). Кроме того, Jupyter Notebook входит в дистрибутив Питона под названием Anaconda.

На сегодняшнем занятии мы рассмотрим именно второй вариант установки.

Anaconda

Anaconda — это дистрибутив Питона и репозиторий пакетов, специально предназначенных для анализа данных и машинного обучения.



Основу дистрибутива Anaconda составляет система управления пакетами и окружениями **conda**.

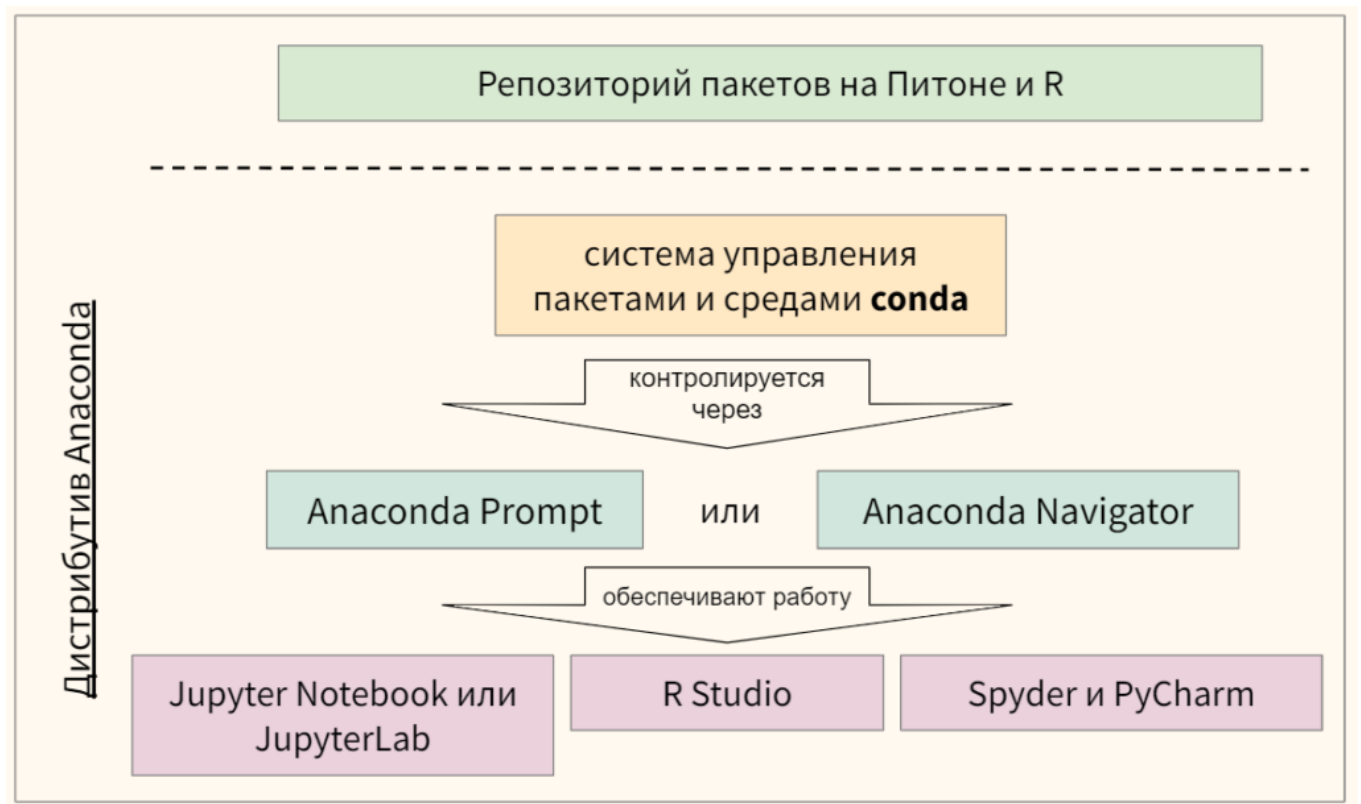
Conda можно управлять двумя способами, а именно через **Anaconda Prompt** — программу, аналогичную командной строке Windows, или через **Anaconda Navigator** — понятный графический интерфейс.

Кроме того, в дистрибутив Anaconda входит несколько полезных программ:

- **Jupyter Notebook** и **JupyterLab** — это программы, позволяющие исполнять код на Питоне (и, как мы увидим, на других языках) и обрабатывать данные.
- **Spyder** и **PyCharm** представляют собой так называемую интегрированную среду разработки (Integrated Development Environment, IDE). IDE — это редактор кода наподобие программы Atom или Sublime Text с дополнительными возможностями автодополнения, компиляции и интерпретации, анализа ошибок, отладки (debugging), подключения к базам данных и др.

- **RStudio** — интегрированная среда разработки для программирования на R.

На схеме структура Anaconda выглядит следующим образом:

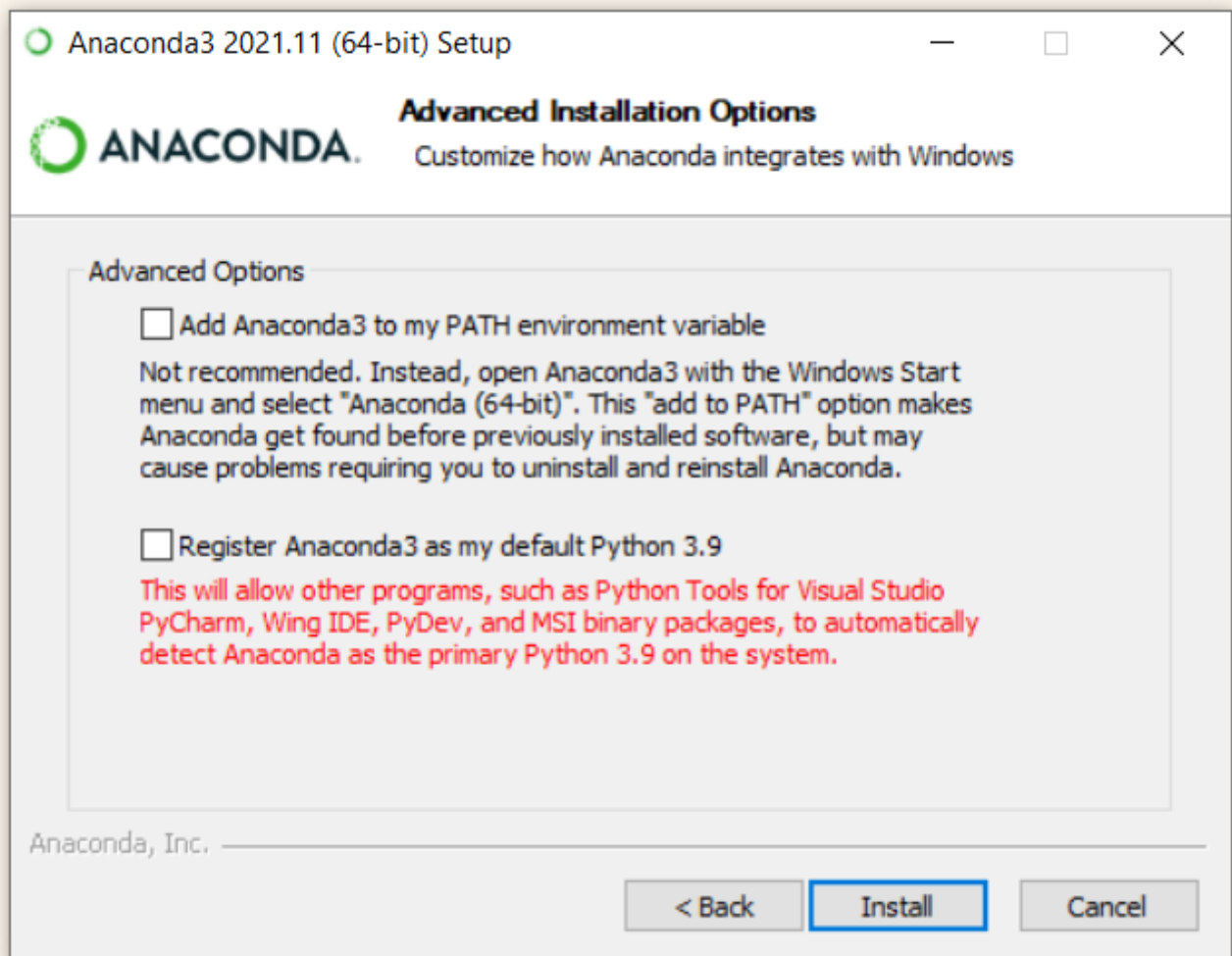


Установка дистрибутива Anaconda на Windows

Шаг 1. Скачайте Anaconda [\[1\]](#) с официального сайта.

Шаг 2. Запустите установщик.

На одном из шагов установки вам предложат поставить две галочки, в частности (1) добавить Anaconda в переменную path и (2) сделать дистрибутив Anaconda версией, которую Windows обнаруживает по умолчанию.



Не отмечайте ни один из пунктов!

Так вы сможете использовать два дистрибутива Питона, первый дистрибутив мы установили на прошлом занятии, второй — сейчас.

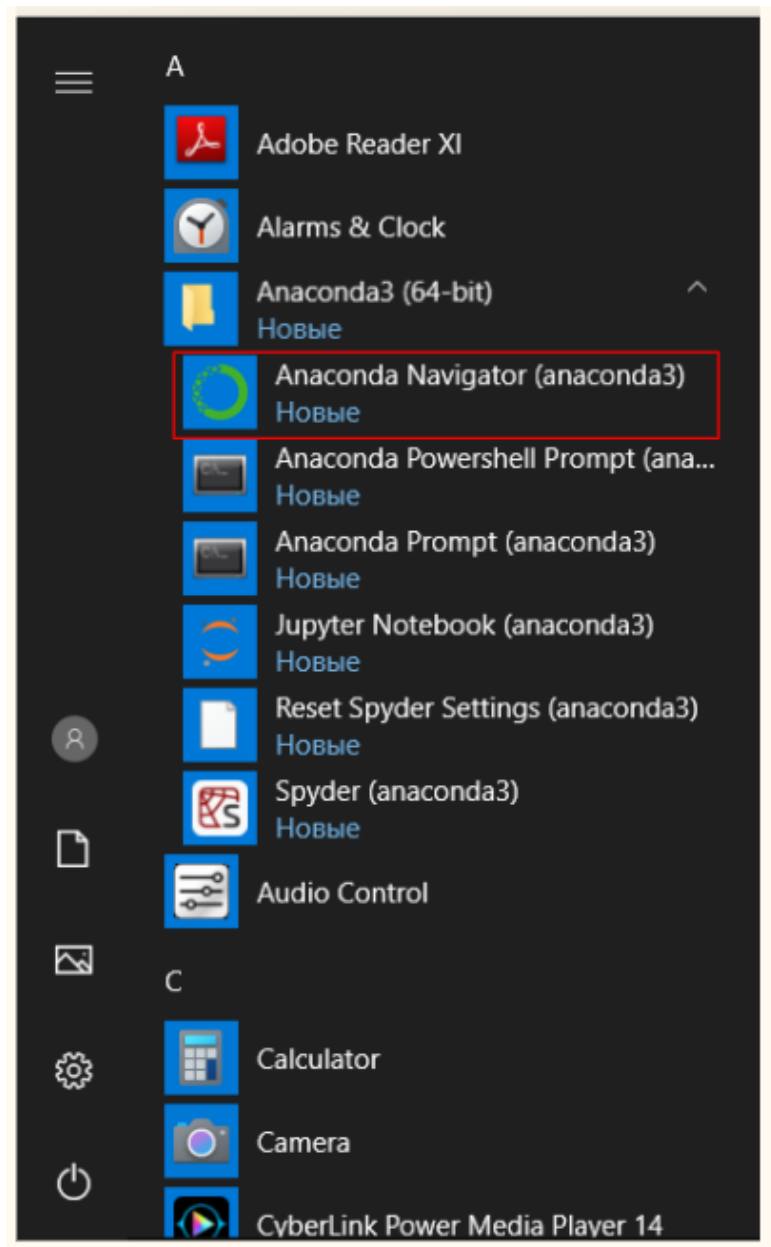
Как запустить Jupyter Notebook

После того как вы скачали и установили Anaconda, можно переходить к запуску ноутбука.

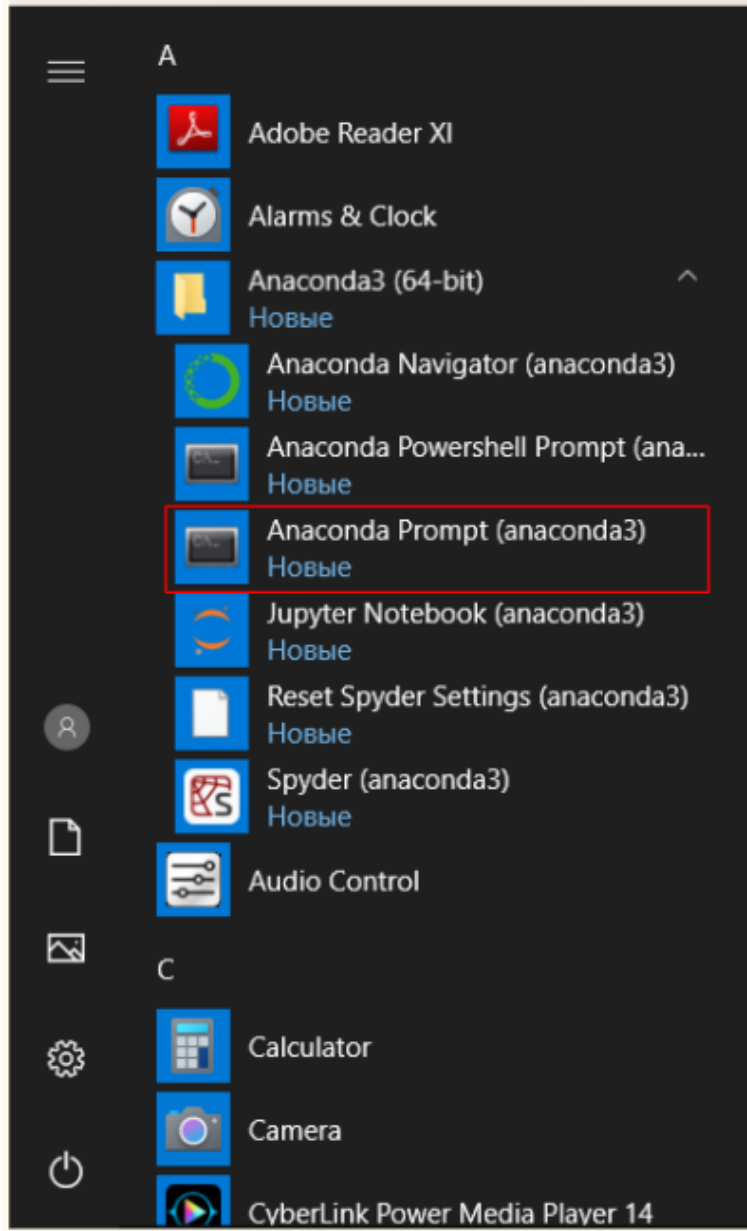
Шаг 1. Откройте Anaconda Navidator

Открыть Anaconda Navigator можно двумя способами.

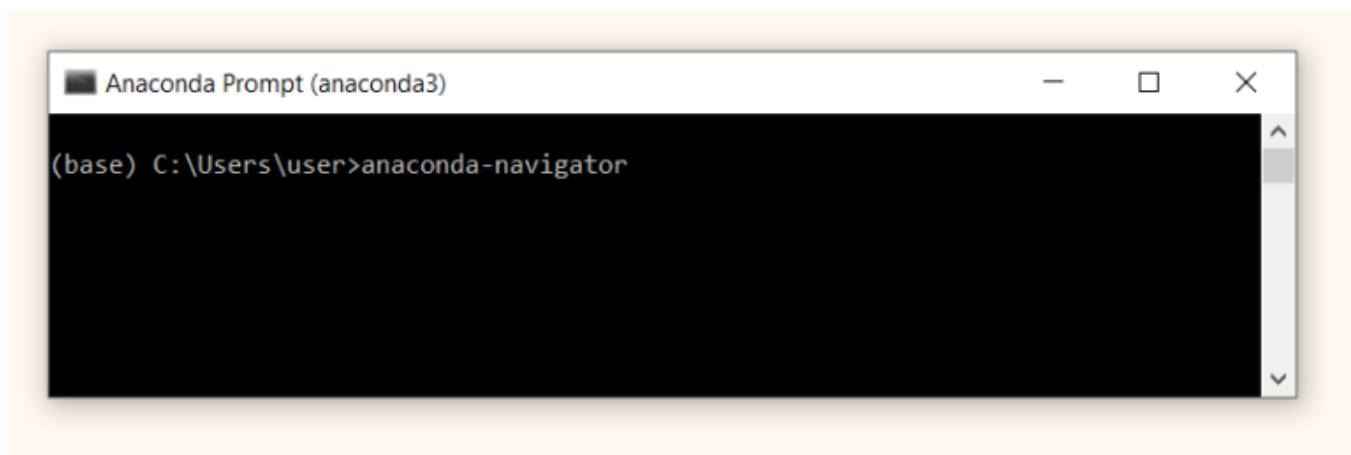
Способ 1. Запуск из меню «Пуск». Просто перейдите в меню «Пуск» и выберите Anaconda Navigator.



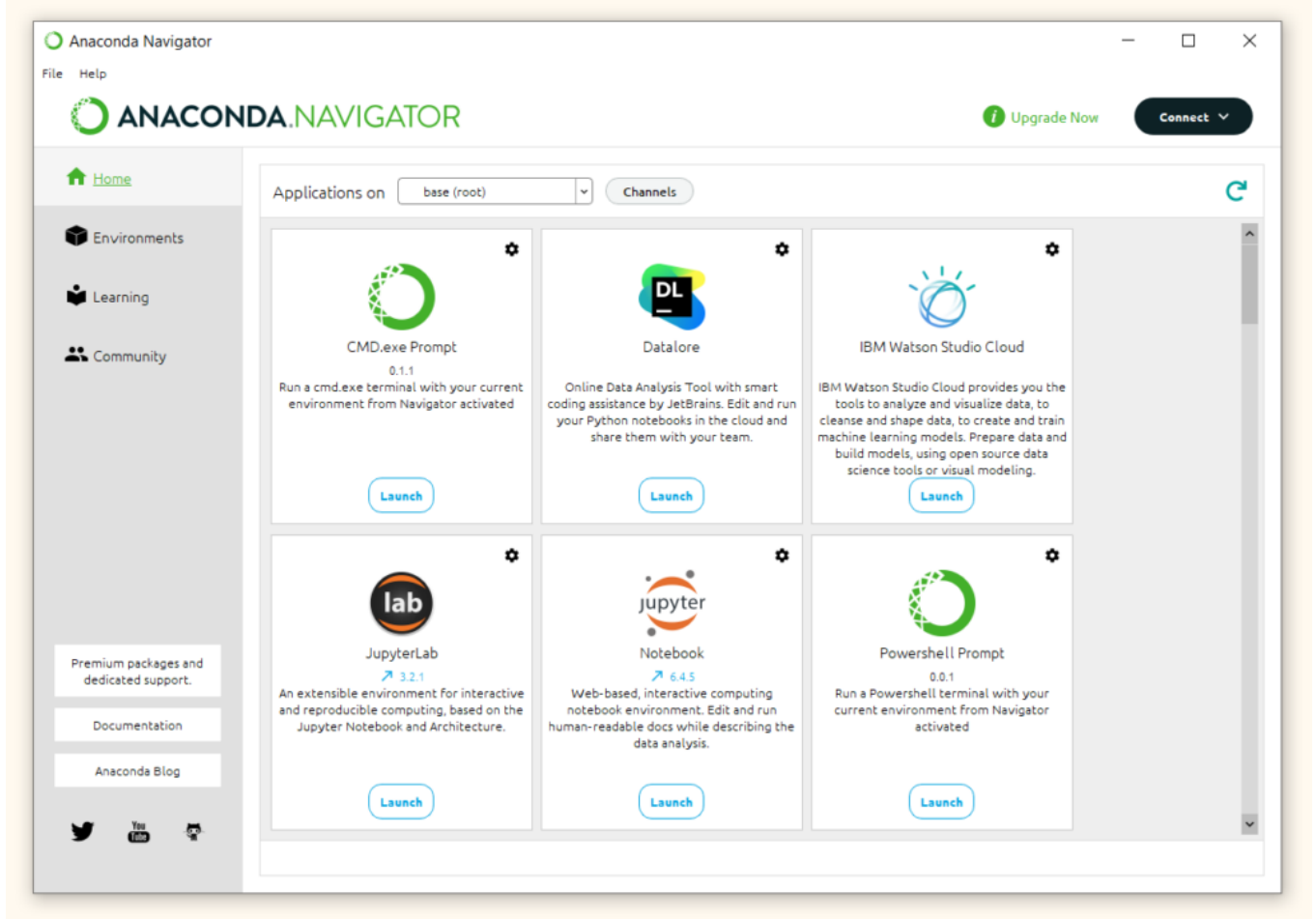
Способ 2. Запуск через Anaconda Prompt. Также из меню «Пуск» откройте терминал Anaconda Prompt.



Введите команду `anaconda-navigator`.

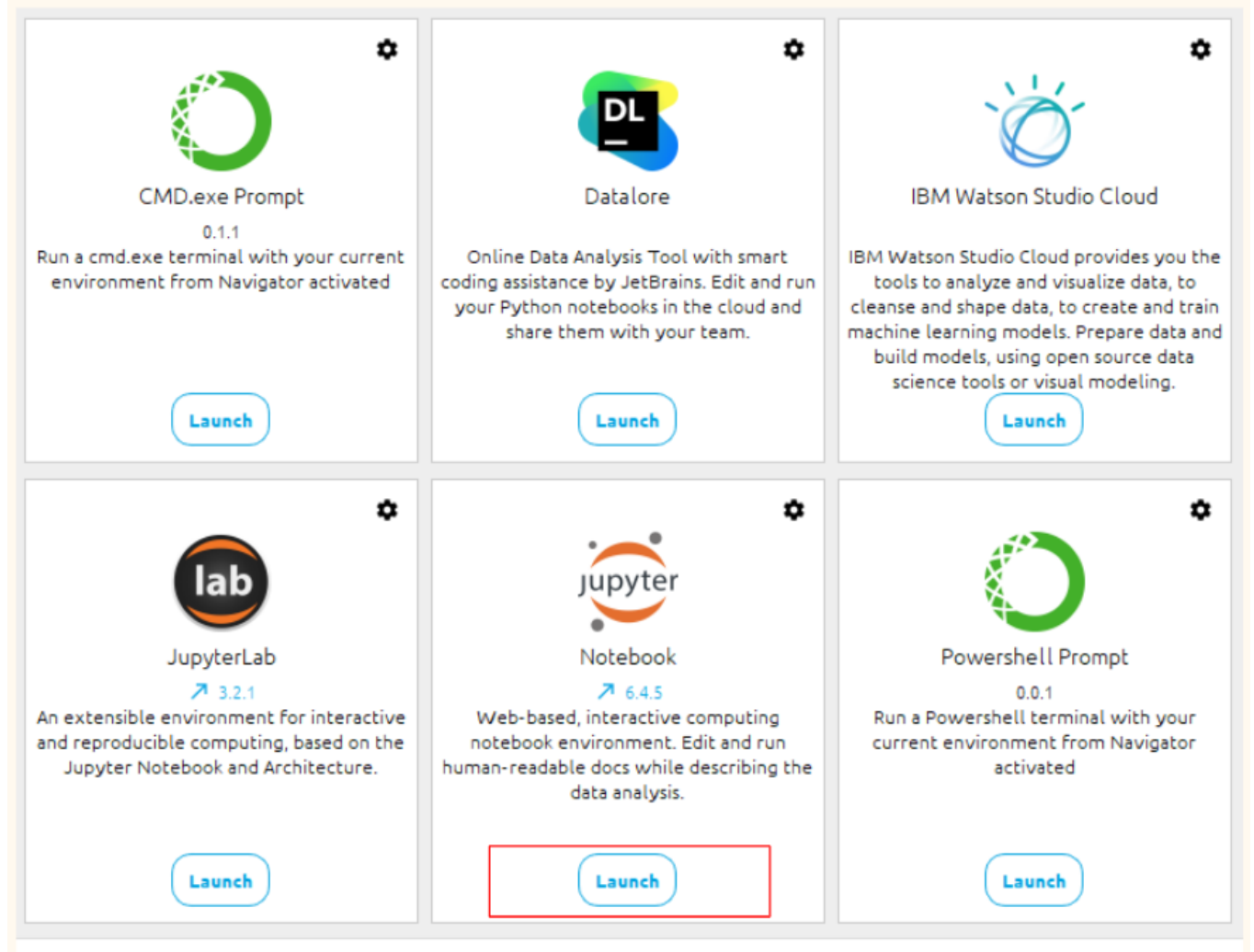


В результате должно появиться вот такое окно.



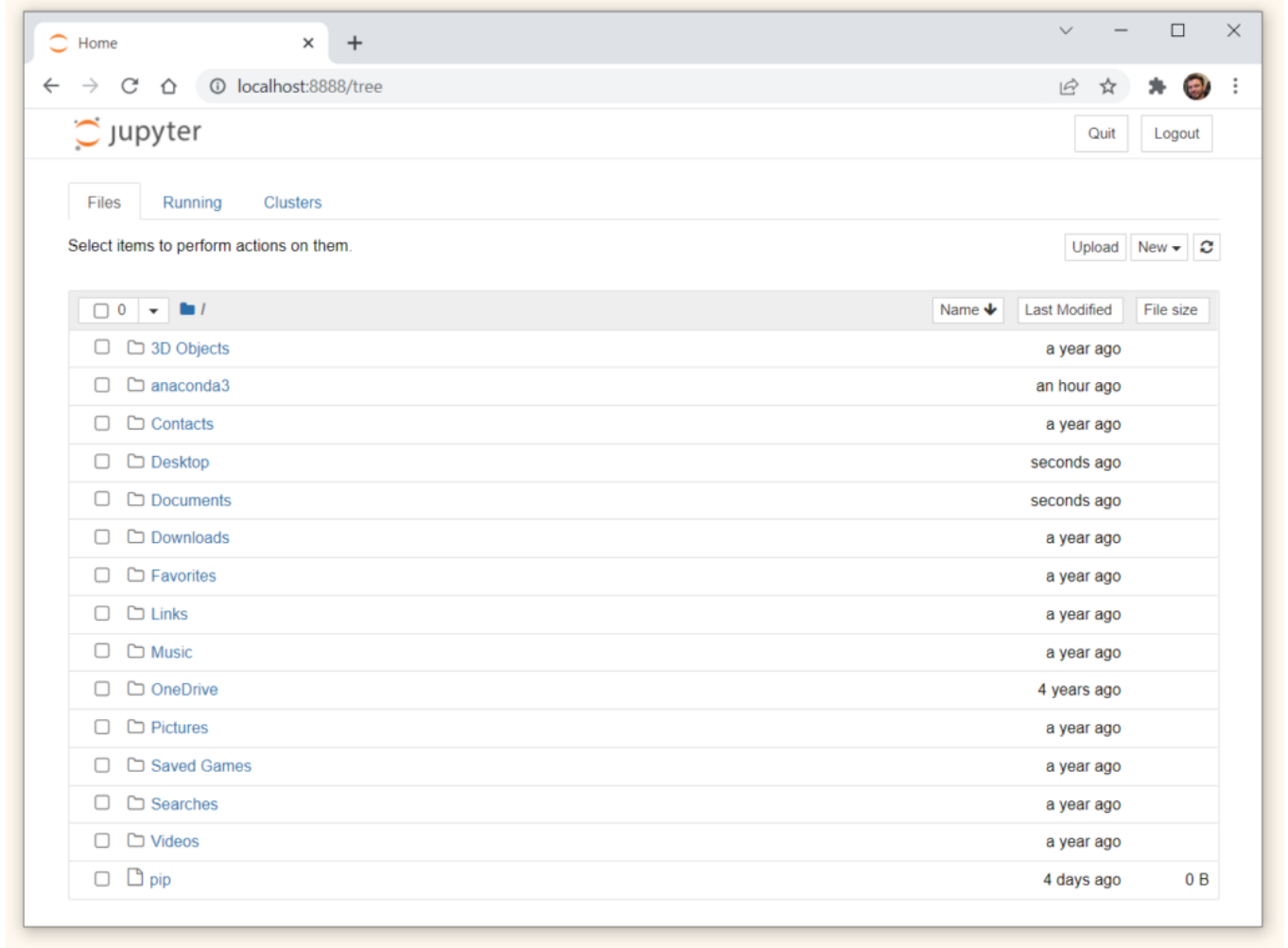
Шаг 2. Откройте Jupyter Notebook

Теперь выберите Jupyter Notebook и нажмите **Launch** («Запустить»).



Замечу, что Jupyter Notebook можно открыть не только из Anaconda Navigator, но и через меню «Пуск», а также введя в терминале Anaconda Prompt команду `jupyter-notebook`.

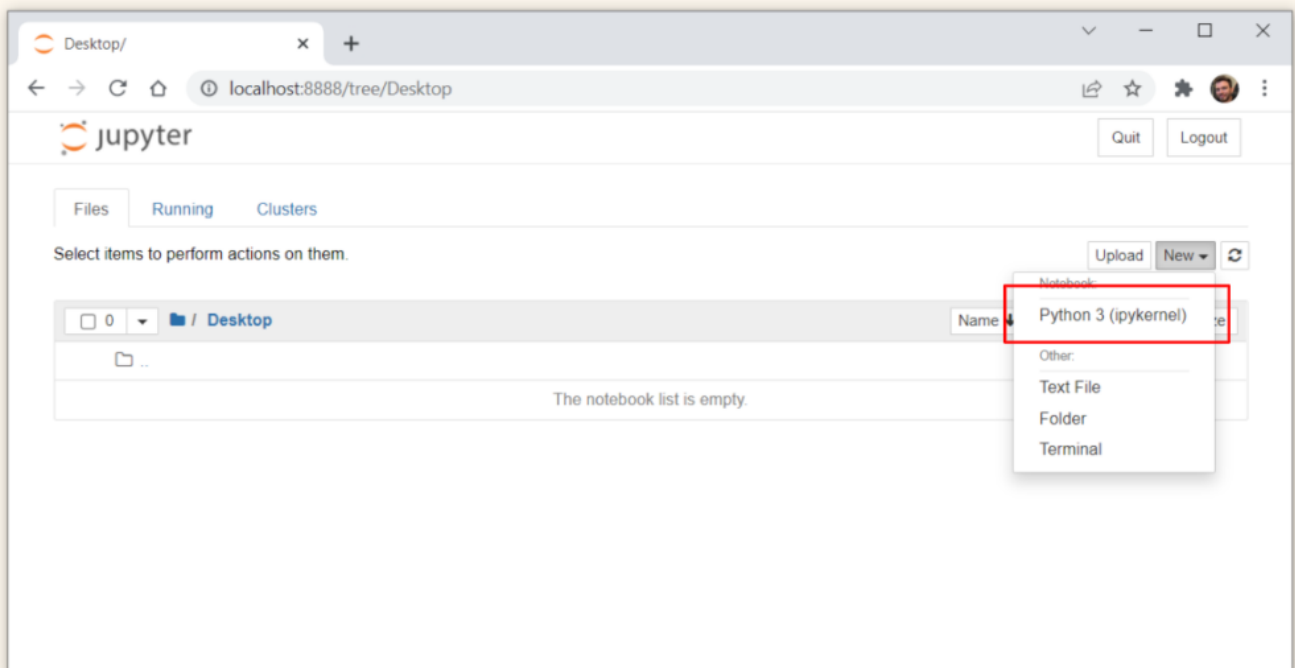
В результате должен запускаться локальный сервер, и в браузере откроется перечень папок вашего компьютера.



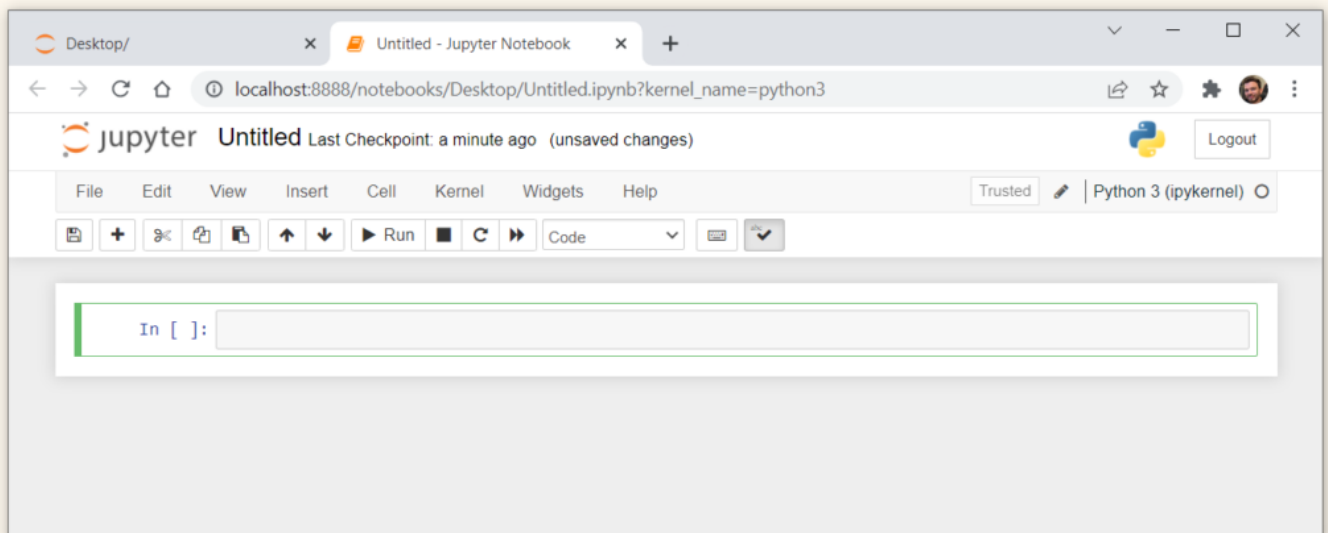
Шаг 3. Выберите папку и создайте ноутбук

Выберите папку, в которой хотите создать ноутбук. В моем случае я выберу **Рабочий стол** (Desktop).

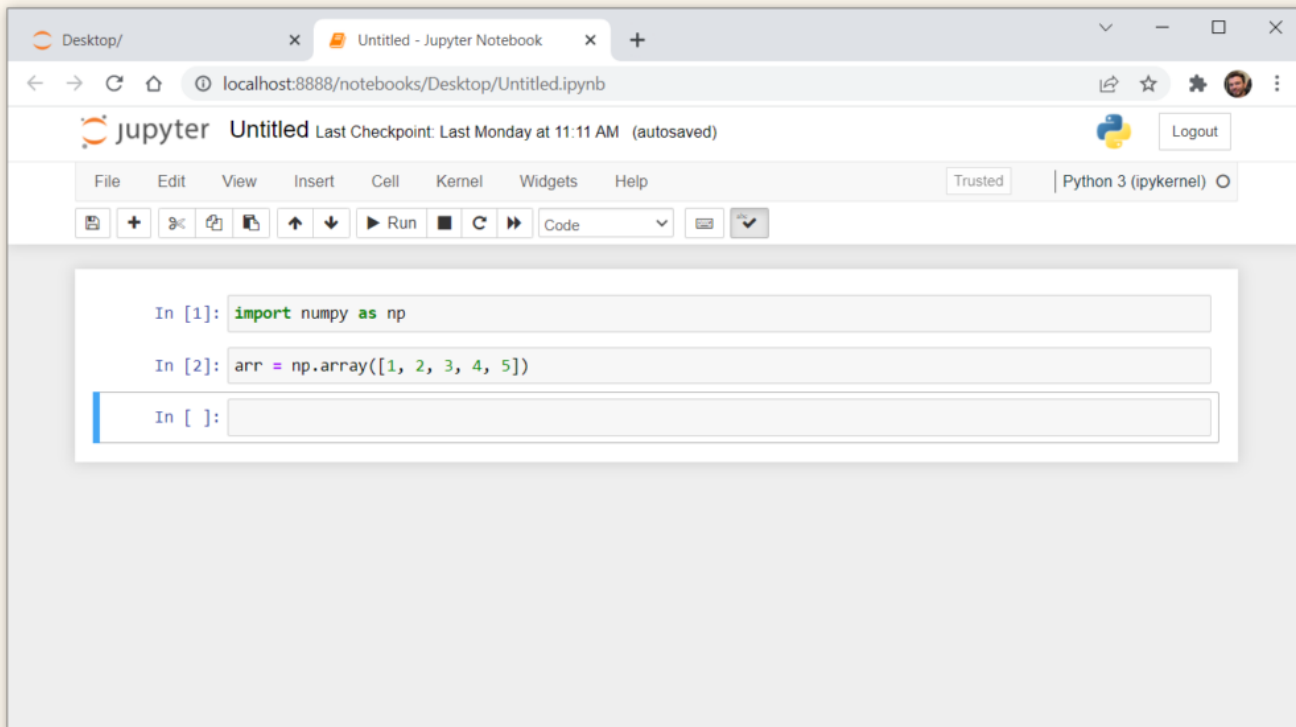
Теперь в правом верхнем углу нажмите **New** → **Python 3**.



Мы готовы писать и исполнять код точно также, как мы это делаем в Google Colab.

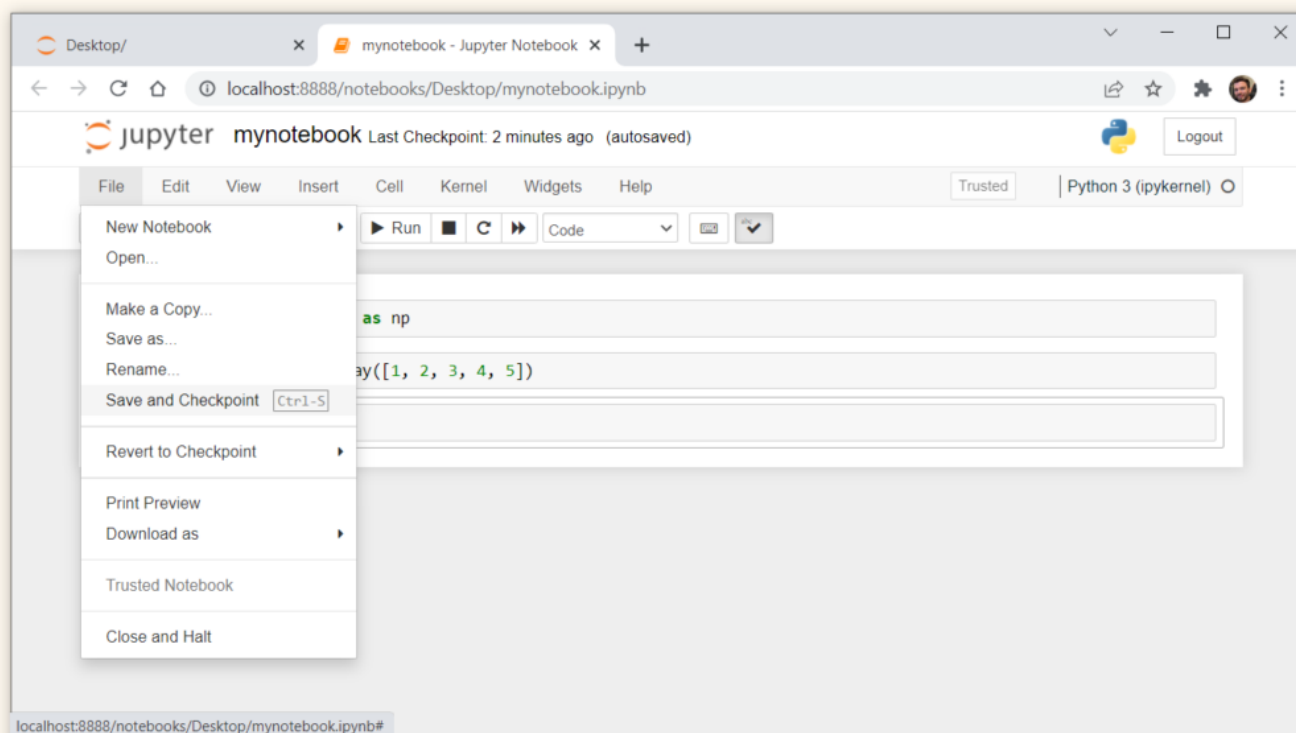


Импортируем библиотеку NumPy и создадим массив.



Шаг 4. Сохраните ноутбук и закройте Jupyter Notebook

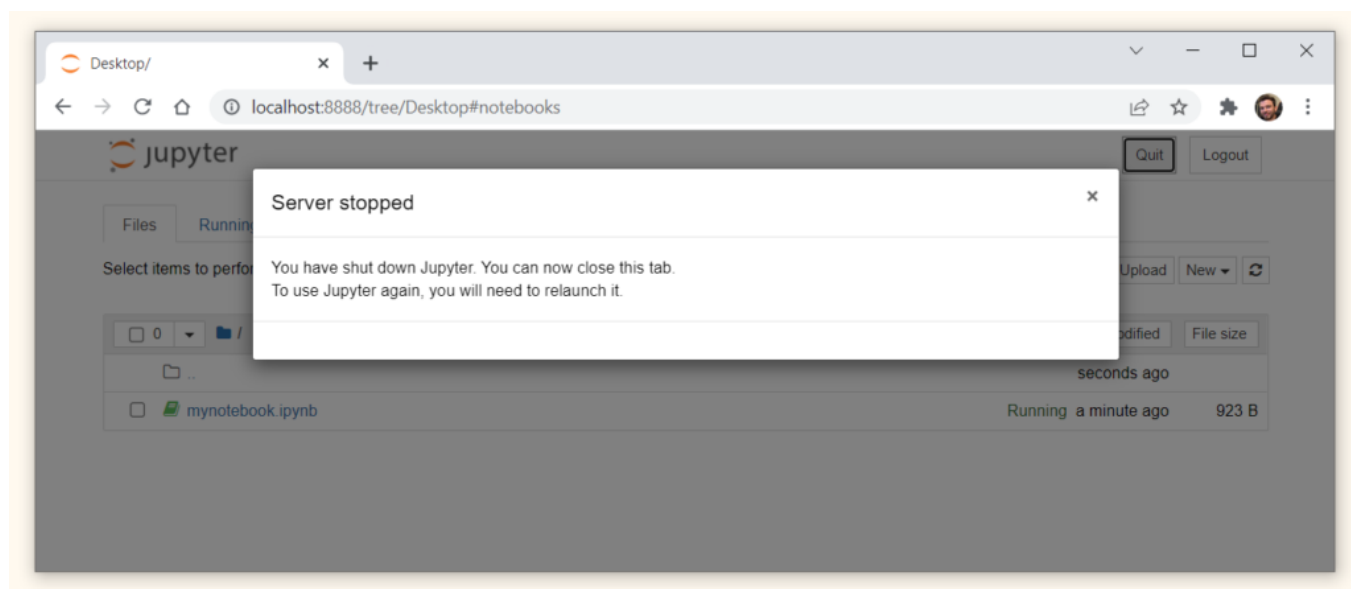
Переименуйте ноутбук в **mynotebook** (для этого, как и в Google Colab, отредактируйте само название непосредственно в окне ноутбука). Сохранить файл можно через **File** → **Save and Checkpoint**.



Обратите внимание, помимо файла `mynotebook.ipynb`, Jupyter Notebook создал скрытую папку `.ipynb_checkpoints`. В ней хранятся файлы, которые позволяют

вернуться к предыдущей сохраненной версии ноутбука (предыдущему checkpoint). Сделать это можно, нажав **File** → **Revert to Checkpoint** и выбрав дату и время предыдущей сохраненной версии кода.

Когда вы закончили работу, закройте вкладку с ноутбуком. Остается прервать работу локального сервера, нажав **Quit** в правом верхнем углу.



Особенности работы

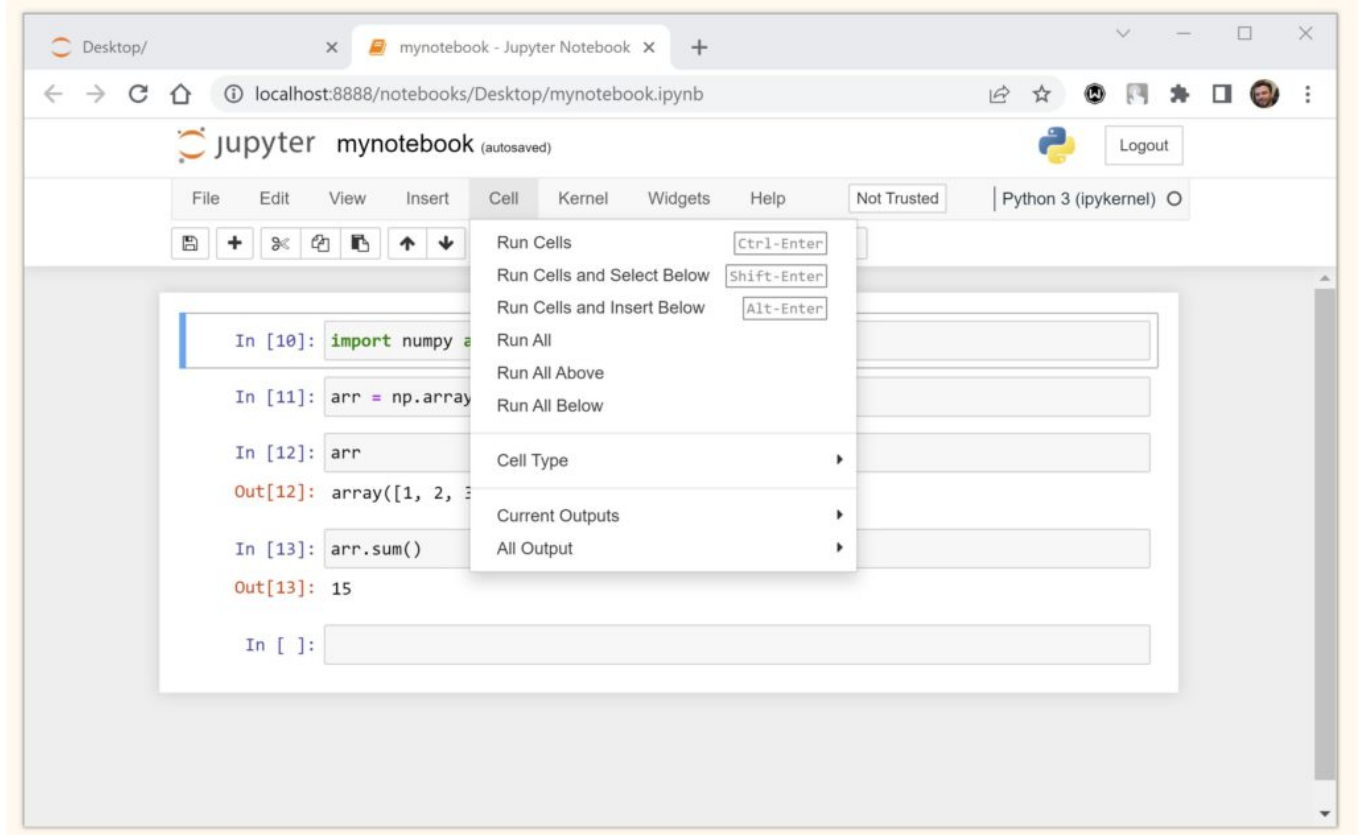
Давайте подробнее поговорим про возможности Jupyter Notebook. Снова запустим только что созданный ноутбук любым удобным способом.

Код на Python

В целом мы пишем обычный код на Питоне.

Вкладка Cell

Для управления запуском или исполнением ячеек можно использовать вкладку Cell.

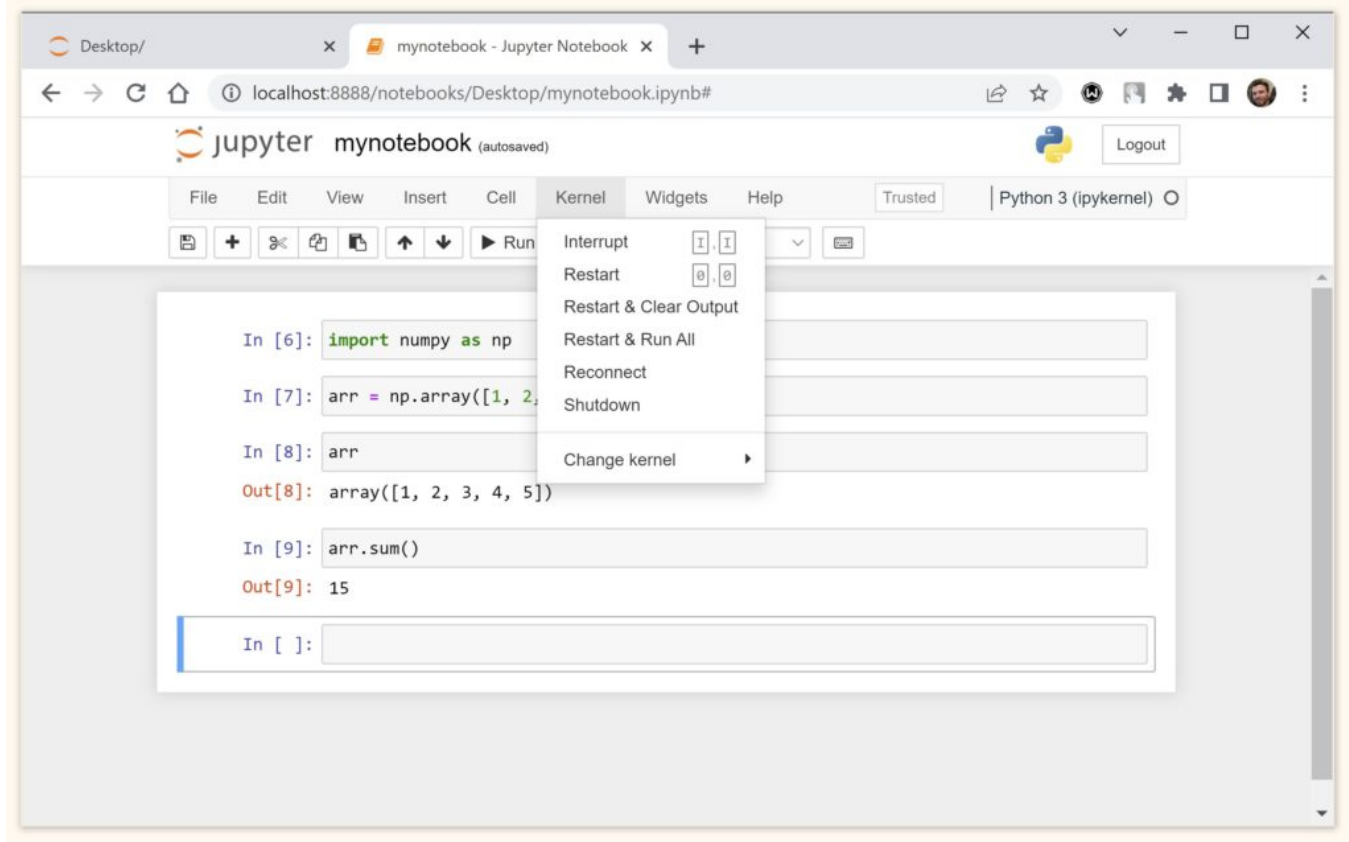


Здесь мы можем, в частности:

- Запускать ячейку и оставаться в ней же через **Run Cells**
- Исполнять все ячейки в ноутбуке, выбрав **Run All**
- Исполнять все ячейки выше (**Run All Above**) или ниже текущей (**Run All Below**)
- Очистить вывод ячеек, нажав **All Output** → **Clear**

Вкладка Kernel

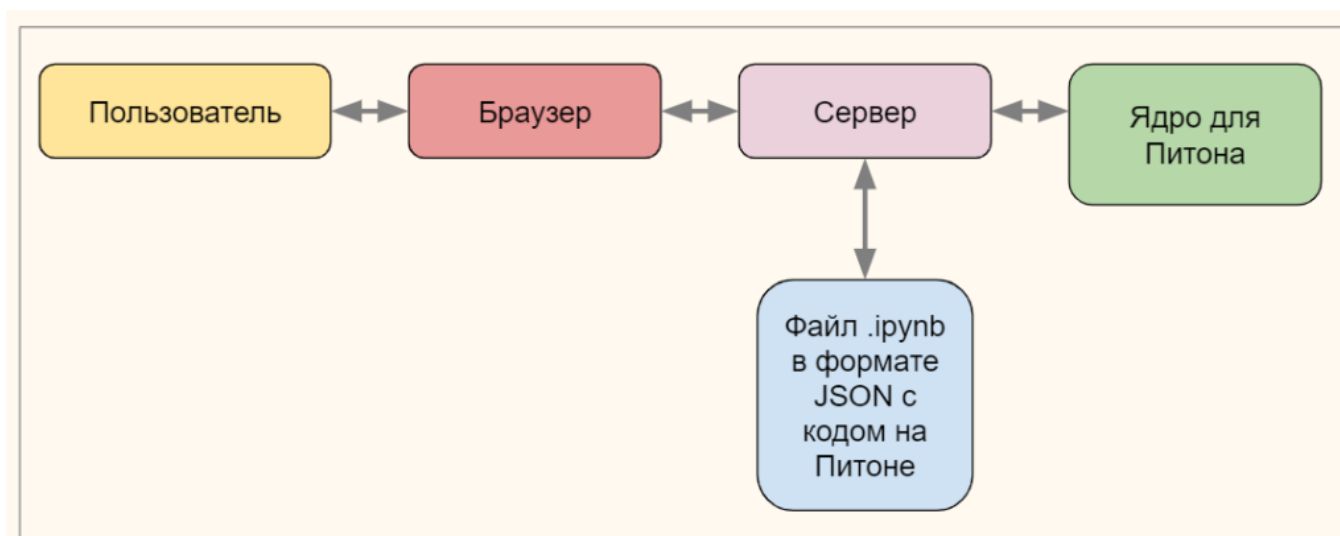
Командами вкладки Kernel мы управляем ядром (kernel) или вычислительным «движком» ноутбука.



В этой вкладке мы можем, в частности:

- Прервать исполнение ячейки командой **Interrupt**. Это бывает полезно, если, например, исполнение кода занимает слишком много времени или в коде есть ошибка и исполнение кода не прервется самостоятельно.
- Перезапустить kernel можно командой **Restart**. Кроме того, можно
 - очистить вывод (**Restart & Clear Output**) и
 - заново запустить все ячейки (**Restart & Run All**)

Несколько слов про то, что такое ядро и как в целом функционирует Jupyter Notebook.



Пользователь взаимодействует с ноутбуком через браузер. Браузер в свою очередь отправляет запросы на сервер. Функция сервера заключается в том, чтобы загружать ноутбук и сохранять внесенные изменения в формате JSON с расширением `.ipynb`. Одновременно, сервер обращается к ядру в тот момент, когда необходимо обработать код на каком-либо языке (например, на Питоне).

Такое «разделение труда» между браузером, сервером и ядром позволяет во-первых, запускать Jupyter Notebook в любой операционной системе, во-вторых, в одной программе исполнять код на нескольких языках, и в-третьих, сохранять результат в файлах одного и того же формата.

Возможность программирования на нескольких языках (а значит использование нескольких ядер) мы изучим чуть позже, а пока посмотрим как устанавливать новые пакеты для Питона внутри Jupyter Notebook.

Установка новых пакетов

Установить новые пакеты в Anaconda можно непосредственно в ячейке, введя `!pip install <package_name>`. Например, попробуем установить Numpy.

```
In [6]: !pip install numpy
```

```
Requirement already satisfied: numpy in c:\users\dmvma\anaconda3\lib\site-packages (1.21.5)
```

Система сообщила нам, что такой пакет уже установлен. Более того, мы видим путь к папке внутри дистрибутива Anaconda, в которой Jupyter «нашел» Numpy.

При подготовке этого занятия я использовал два компьютера, поэтому имя пользователя на скриншотах указано как user или dmvma. На вашем компьютере при указании пути к файлу используйте ваше имя пользователя.

В последующих разделах мы рассмотрим дополнительные возможности по установке пакетов через Anaconda Prompt и Anaconda Navigator.

По ссылке ниже вы можете скачать код, который мы создали в Jupyter Notebook.

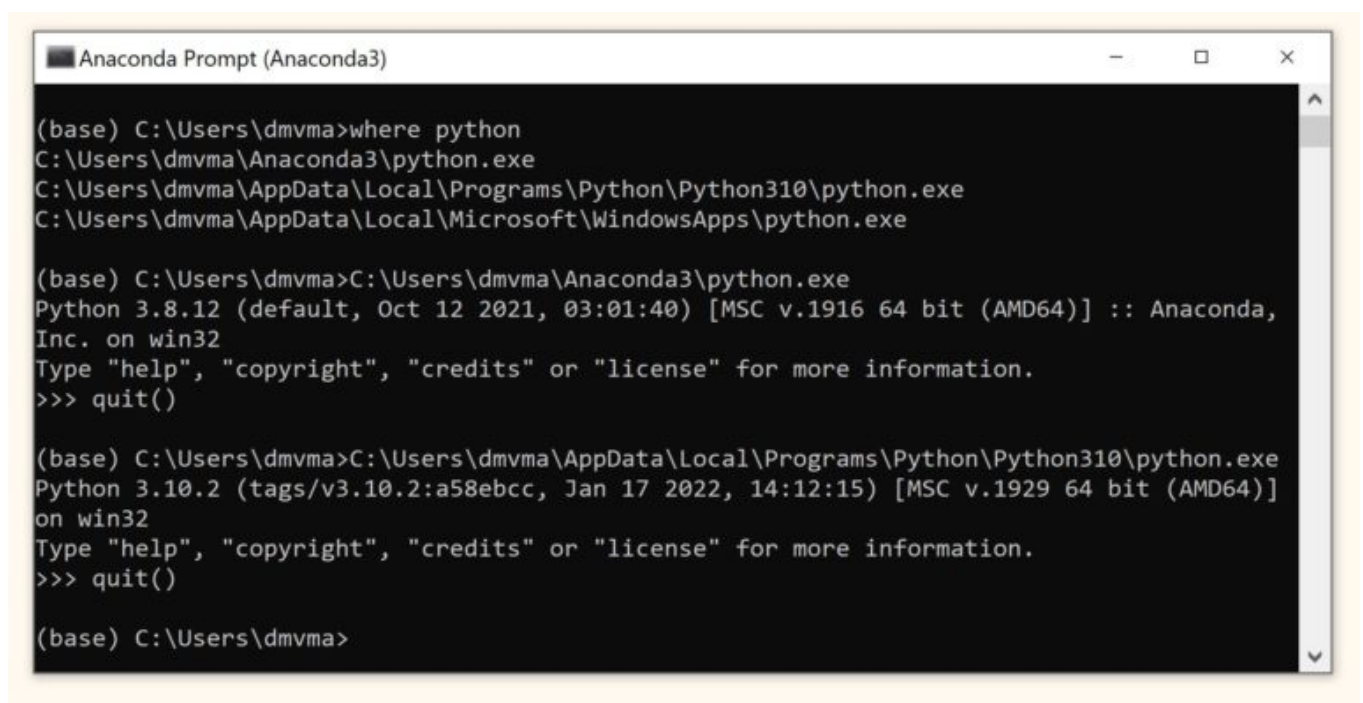
mynotebook.zip

Скачать

Два Питона на одном компьютере

Обращу ваше внимание, что на данный момент на моем компьютере (как и у вас, если вы проделали шаги прошлого занятия) установлено два Питона, один с сайта www.python.org, второй — в составе дистрибутива Anaconda.

Посмотреть на установленные на компьютеры «Питоны» можно, набрав команду `where python` в Anaconda Prompt.



```
Anaconda Prompt (Anaconda3)

(base) C:\Users\dmvma>where python
C:\Users\dmvma\Anaconda3\python.exe
C:\Users\dmvma\AppData\Local\Programs\Python\Python310\python.exe
C:\Users\dmvma\AppData\Local\Microsoft\WindowsApps\python.exe

(base) C:\Users\dmvma>C:\Users\dmvma\Anaconda3\python.exe
Python 3.8.12 (default, Oct 12 2021, 03:01:40) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> quit()

(base) C:\Users\dmvma>C:\Users\dmvma\AppData\Local\Programs\Python\Python310\python.exe
Python 3.10.2 (tags/v3.10.2:a58ebcc, Jan 17 2022, 14:12:15) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> quit()

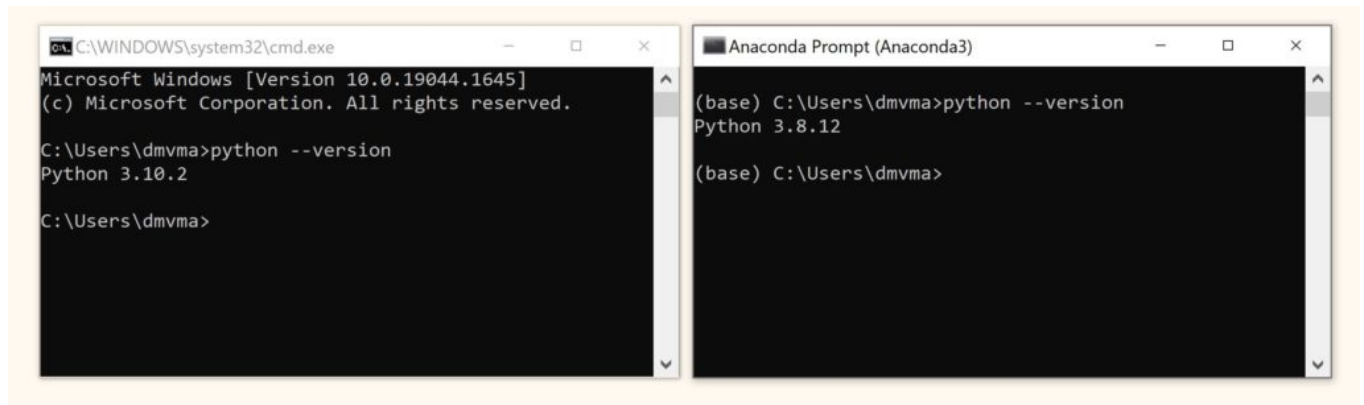
(base) C:\Users\dmvma>
```

Указав полный или абсолютный путь (absolute path) к каждому из файлов `python.exe`, мы можем в интерактивном режиме исполнять код на версии 3.8 (установили с www.python.org) и на версии 3.10 (установили в составе Anaconda). При запуске файла `python.exe` из папки `WindowsApps` система предложит установить Питон из Microsoft Store.

В этом смысле нужно быть аккуратным и понимать, какой именно Питон вы используете и куда устанавливаете очередной пакет.

В нашем случае мы настроили работу так, чтобы устанавливать библиотеки для Питона с www.python.org через командную строку Windows, и устанавливать пакеты в Анаконду через Anaconda Prompt.

Убедиться в этом можно, проверив версии Питона через `python --version` в обеих программах.



Теперь попробуйте ввести в них команду `pip list` и сравнить установленные библиотеки.

Markdown в Jupyter Notebook

Вернемся к Jupyter Notebook. Помимо ячеек с кодом, можно использовать текстовые ячейки, в которых поддерживается язык разметки Markdown. Мы уже коротко рассмотрели этот язык на прошлом занятии, когда создавали пакет на Питоне.

По большому счету, с помощью несложных команд Markdown, вы говорите Jupyter как отформатировать ту или иную часть текста.

Рассмотрим несколько основных возможностей форматирования (для удобства и в силу практически полного совпадения два последующих раздела приведены в ноутбуке Google Colab).

Откроем ноутбук к этому занятию 

Заголовки

Заголовки создаются с помощью символа решетки.

```
# Заголовок 1
## Заголовок 2
### Заголовок 3
#### Заголовок 4
##### Заголовок 5
##### Заголовок 6
```

Заголовок 1

Заголовок 2

Заголовок 3

Заголовок 4

Заголовок 5

Заголовок 6

Если перед первым символом решетки поставить знак \, Markdown просто выведет символы решетки.

```
\ # Заголовок 1
\ ## Заголовок 2
\ ### Заголовок 3
\ #### Заголовок 4
\ ##### Заголовок 5
\ ##### Заголовок 6
```

Абзацы

Абзацы отделяются друг от друга пробелами.

```
Абзац 1
```

Абзац 2

Абзац 1

Абзац 2

Мы также можем разделять абзацы прямой линией.

Выделение текста

****Полужирный стиль****

Курсив

~~Перечеркнутый стиль~~

Полужирный стиль

Курсив

Перечеркнутый стиль

Форматирование кода и выделенные абзацы

Мы можем выделять код внутри строки или отдельным абзацем.

```
Отформатируем код `print('Hello world!')` внутри строки и отдельным абзацем
...
print('Hello world!')
...
```

Отформатируем код `print('hello world!')` внутри строки и отдельным абзацем

```
print('Hello world!')
```

Возможно выделение и текстовых абзацев (так называемые blockquotes).

```
> Markdown позволяет форматировать текст без использования тэгов.  
>  
> Он был создан в 2004 году Джоном Грубером и Аароном Шварцем.
```

Markdown позволяет форматировать текст без использования тэгов.

Он был создан в 2004 году Джоном Грубером и Аароном Шварцем.

Списки

Посмотрим на создание упорядоченных и неупорядоченных списков.

****Упорядоченный список****

1. Пункт 1

1. Пункт 2 (нумерация ведется автоматически)

****Неупорядоченный список****

* Пункт 1.1

* Пункт 2.1

* Пункт 2.2

* Пункт 3.1

* Пункт 3.2

* Пункт 1.2

Упорядоченный список

1. Пункт 1
2. Пункт 2 (нумерация ведется автоматически)

Неупорядоченный список

- Пункт 1.1
 - Пункт 2.1
 - Пункт 2.2
 - Пункт 3.1
 - Пункт 3.2
- Пункт 1.2

Ссылки и изображения

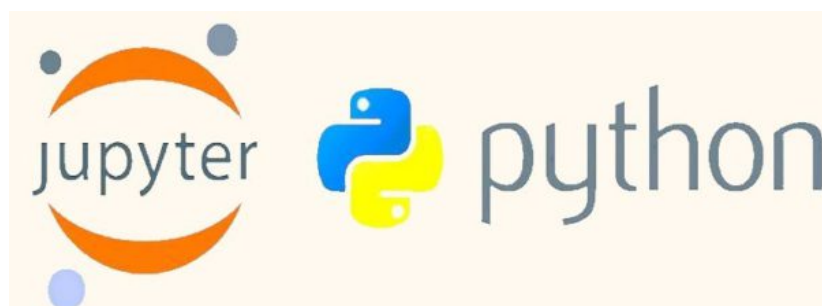
Текст ссылки заключается в квадратные скобки, сама ссылка — в круглые.

```
[сайт проекта Jupyter](https://jupyter.org/)
```

[сайт проекта Jupyter](https://jupyter.org/)

Изображение форматируется похожим образом.

```
![логотипы Jupyter и Python](https://www.dmitrymakarov.ru/wp-content/uploa
```



Таблицы

id	item	price
01	pen	200

	02 pencil 150
	03 notebook 300

id	item	price
01	pen	200
02	pencil	150
03	notebook	300

Таблицы для Markdown бывает удобно создавать с помощью специального инструмента⁴.

Формулы на LaTeX

В текстовых полях можно вставлять формулы и математические символы с помощью системы верстки, которая называется LaTeX (произносится «латэк»). Они заключаются в одинарные или двойные символы \$.

Если использовать одинарный символ \$, то расположенная внутри формула останется в пределах того же абзаца. Например, запись `$ y = x^2 $` даст $y = x^2$.

В то время как `$$ y = x^2 $$` поместит формулу в новый абзац.

$$y = x^2$$

Одинарный символ \ добавляет пробел. Двойной символ \ переводит текст на новую строку.

```
$$ \hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n
```

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$$

Рассмотрим некоторые элементы синтаксиса LaTeX.

Форматирование текста

```
1 $ \text{just text} $
2
3 $ \textbf{bold} $
4
5 $ \textit{italic} $
6
7 $ \underline{underline} $
```

just text

bold

italic

underline

Надстрочные и подстрочные знаки

```
1 hat $ \hat{x} $
2
3 bar $ \bar{x} $
4
5 vector $ \vec{x} $
6
7 tilde $ \tilde{x} $
8
9 superscript $ e^{ax + b} $
10
11 subscript $ A_{i, j} $
12
13 degree $ 90^{\circ} $
```

hat \hat{x}

bar \bar{x}

vector \vec{x}

tilde \tilde{x}

superscript e^{ax+b}

subscript $A_{i,j}$

degree 90°

Скобки

Вначале рассмотрим код для скобок в пределах высоты строки.

```
1  $$
2  (a+b) \\
3  [a+b] \\
4  \{a+b\} \\
5  \langle x+y \rangle \\
6  |x+y| \\
7  \|x+y\|
8  $$
```

$$\begin{aligned} &(a + b) \\ &[a + b] \\ &\{a + b\} \\ &\langle x + y \rangle \\ &|x + y| \\ &\|x + y\| \end{aligned}$$

Кроме того, с помощью `\left(`, `\right)`, а также `\left[`, `\right]` и так далее можно увеличить высоту скобки. Сравните.

```
1  $$
2  \left(\frac{1}{2}\right) \quad \left(\frac{1}{2}\right)
3  $$
```

$$\left(\frac{1}{2}\right) \quad \left(\frac{1}{2}\right)$$

Также можно использовать отдельные команды для скобок различного размера.

```
1  $$
2  \big( \Big( \bigg( \Bigg( \\
3  \big] \Big] \bigg] \Bigg] \\
4  \big\{ \Big\{ \bigg\{ \Bigg\{
5  $$
```

((((
]]]]
{}}{

Дробь и квадратный корень

```
1 fraction
2
3 $$ \frac{1}{1+e^{-z}} $$
4
5 square root $ \sqrt{\sigma^2} $
```

fraction

$$\frac{1}{1 + e^{-z}}$$

square root $\sqrt{\sigma^2}$

Греческие буквы

1	Uppercase	LaTeX	Lowercase	LaTeX	RU	
2	-----	-----	-----	-----	-----	
3	-----	-----	\$\alpha\$	\alpha	альфа	
4	-----	-----	\$\beta\$	\beta	бета	
5	\$\Gamma\$	\Gamma	\$\gamma\$	\gamma	гамма	
6	\$\Delta\$	\Delta	\$\delta\$	\delta	дельта	
7	-----	-----	\$\epsilon\$	\epsilon	эпсилон	
8	-----	-----	\$\varepsilon\$	\varepsilon	-----	
9	-----	-----	\$\zeta\$	\zeta	дзета	
10	-----	-----	\$\eta\$	\eta	эта	
11	\$\Theta\$	\Theta	\$\theta\$	\theta	тета	
12	-----	-----	\$\vartheta\$	\vartheta	-----	
13	-----	-----	\$\iota\$	\iota	йота	
14	-----	-----	\$\kappa\$	\kappa	каппа	
15	\$\Lambda\$	\Lambda	\$\lambda\$	\lambda	лямбда	

16	-----	-----	\mu\$	\\mu		мю	
17	-----	-----	\nu\$	\\nu		ню	
18	\Xi\$	\\Xi	\xi\$	\\xi		кси	
19	-----	-----	\omicron\$	\\omicron		омикрон	
20	\Pi\$	\\Pi	\pi\$	\\pi		пи	
21	-----	-----	\varpi\$	\\varpi		-----	
22	-----	-----	\rho\$	\\rho		ро	
23	-----	-----	\varrho\$	\\varrho		-----	
24	\Sigma\$	\\Sigma	\sigma\$	\\sigma		сигма	
25	-----	-----	\varsigma\$	\\varsigma		-----	
26	-----	-----	\tau\$	\\tau		тау	
27	\Upsilon\$	\\Upsilon	\upsilon\$	\\upsilon		ипсилон	
28	\Phi\$	\\Phi	\phi\$	\\phi		фи	
29	-----	-----	\varphi\$	\\varphi		-----	
30	-----	-----	\chi\$	\\chi		хи	
31	\Psi\$	\\Psi	\psi\$	\\psi		пси	
32	\Omega\$	\\Omega	\omega\$	\\omega		омега	

Uppercase	LaTeX	Lowercase	LaTeX	RU
-----	-----	α	<code>\alpha</code>	альфа
-----	-----	β	<code>\beta</code>	бета
Γ	<code>\Gamma</code>	γ	<code>\gamma</code>	гамма
Δ	<code>\Delta</code>	δ	<code>\delta</code>	дельта
-----	-----	ϵ	<code>\epsilon</code>	эпсилон
-----	-----	ε	<code>\varepsilon</code>	-----
-----	-----	ζ	<code>\zeta</code>	дзета
-----	-----	η	<code>\eta</code>	эта
Θ	<code>\Theta</code>	θ	<code>\theta</code>	тета
-----	-----	ϑ	<code>\vartheta</code>	-----
-----	-----	ι	<code>\iota</code>	йота
-----	-----	κ	<code>\kappa</code>	каппа
Λ	<code>\Lambda</code>	λ	<code>\lambda</code>	лямбда
-----	-----	μ	<code>\mu</code>	мю
-----	-----	ν	<code>\nu</code>	ню
Ξ	<code>\Xi</code>	ξ	<code>\xi</code>	кси
-----	-----	\omicron	<code>\omicron</code>	омикрон
Π	<code>\Pi</code>	π	<code>\pi</code>	пи
-----	-----	ϖ	<code>\varpi</code>	-----
-----	-----	ρ	<code>\rho</code>	ро
-----	-----	ϱ	<code>\varrho</code>	-----
Σ	<code>\Sigma</code>	σ	<code>\sigma</code>	сигма
-----	-----	ς	<code>\varsigma</code>	-----
-----	-----	τ	<code>\tau</code>	тау
Υ	<code>\Upsilon</code>	υ	<code>\upsilon</code>	ипсилон
Φ	<code>\Phi</code>	ϕ	<code>\phi</code>	фи
-----	-----	φ	<code>\varphi</code>	-----
-----	-----	χ	<code>\chi</code>	хи
Ψ	<code>\Psi</code>	ψ	<code>\psi</code>	пси
Ω	<code>\Omega</code>	ω	<code>\omega</code>	омега

Латинские обозначения

1	\$\$
2	<code>\sin(-\alpha) = -\sin(\alpha) \\\</code>
3	<code>\cos(\theta)=\sin \left(\frac{\pi}{2}-\theta \right) \\\</code>
4	<code>\tan(x) = \frac{\sin(x)}{\cos(x)} \\\</code>
5	<code>\log_b(1) = 0 \\\</code>
6	\$\$

$$\sin(-\alpha) = -\sin(\alpha)$$
$$\cos(\theta) = \sin\left(\frac{\pi}{2} - \theta\right)$$
$$\tan(x) = \frac{\sin(x)}{\cos(x)}$$
$$\log_b(1) = 0$$

Логические символы и символы множества

1	LaTeX	symbol	
2	-----	-----	
3	\Rightarrow	\$ \Rightarrow \$	
4	\rightarrow	\$ \rightarrow \$	
5	\longleftarrow	\$ \Leftarrow \$	
6	\cap	\$ \cap \$	
7	\cup	\$ \cup \$	
8	\subset	\$ \subset \$	
9	\in	\$ \in \$	
10	\notin	\$ \notin \$	
11	\varnothing	\$ \varnothing \$	
12	\neg	\$ \neg \$	
13	\forall	\$ \forall \$	
14	\exists	\$ \exists \$	
15	\mathbb{N}	\$ \mathbb{N} \$	
16	\mathbb{Z}	\$ \mathbb{Z} \$	
17	\mathbb{Q}	\$ \mathbb{Q} \$	
18	\mathbb{R}	\$ \mathbb{R} \$	
19	\mathbb{C}	\$ \mathbb{C} \$	

LaTeX	symbol
<code>\Rightarrow</code>	\Rightarrow
<code>\rightarrow</code>	\rightarrow
<code>\longleftrightarrow</code>	\Leftrightarrow
<code>\cap</code>	\cap
<code>\cup</code>	\cup
<code>\subset</code>	\subset
<code>\in</code>	\in
<code>\notin</code>	\notin
<code>\varnothing</code>	\emptyset
<code>\neg</code>	\neg
<code>\forall</code>	\forall
<code>\exists</code>	\exists
<code>\mathbb{N}</code>	\mathbb{N}
<code>\mathbb{Z}</code>	\mathbb{Z}
<code>\mathbb{Q}</code>	\mathbb{Q}
<code>\mathbb{R}</code>	\mathbb{R}
<code>\mathbb{C}</code>	\mathbb{C}

Другие символы

1	LaTeX	symbol
2	-----	-----
3	<code><</code>	$\$ < \$$
4	<code>\leq</code>	$\$ \leq \$$
5	<code>></code>	$\$ \geq \$$
6	<code>\neq</code>	$\$ \neq \$$
7	<code>\approx</code>	$\$ \approx \$$
8	<code>\angle</code>	$\$ \angle \$$
9	<code>\parallel</code>	$\$ \parallel \$$
10	<code>\pm</code>	$\$ \pm \$$
11	<code>\mp</code>	$\$ \mp \$$
12	<code>\cdot</code>	$\$ \cdot \$$
13	<code>\times</code>	$\$ \times \$$
14	<code>\div</code>	$\$ \div \$$

LaTeX	symbol
<	<
\leq	≤
>	≥
\neq	≠
\approx	≈
\angle	∠
\parallel	∥
\pm	±
\mp	∓
\cdot	·
\times	×
\div	÷

Кусочная функция и система уравнений

Посмотрим на запись функции sgn (sign function) средствами LaTeX.

```

1  $$
2  \operatorname{sgn}(x) = \left\{ \begin{array}{ll}
3      1 & \text{if } x \in \mathbf{N}^* \\
4      0 & \text{if } x = 0 \\
5      -1 & \text{else.} \\
6  \end{array} \right.
7  \operatorname{sgn}(x) = \left\{ \begin{array}{ll}
8      1 & \text{if } x \in \mathbf{N}^* \\
9      0 & \text{if } x = 0 \\
10     -1 & \text{else.} \\
11 \end{array} \right.
12  $$

```

$$\operatorname{sgn}(x) = \begin{cases} 1 & \text{if } x \in \mathbf{N}^* \\ 0 & \text{if } x = 0 \\ -1 & \text{else.} \end{cases}$$

Схожим образом записывается система линейных уравнений.

```

1  $$
2  \left\{ \begin{array}{l}

```

```

3 \begin{matrix}
4 4x + 3y = 20 \\
5 -5x + 9y = 26
6 \end{matrix}
7 \right.
8 $$

```

$$\left\{ \begin{array}{l} 4x + 3y = 20 \\ -5x + 9y = 26 \end{array} \right.$$

Горизонтальная фигурная скобка

```

1 $$
2 \overbrace{
3 \underbrace{a}_{\text{real}} +
4 \underbrace{b}_{\text{imaginary}} i}
5 ^{\text{complex number}}
6 $$

```

$$\overbrace{\underbrace{a}_{\text{real}} + \underbrace{b}_{\text{imaginary}} i}^{\text{complex number}}$$

Предел, производная, интеграл

```

1 Пределы:
2
3 $$ \lim_{x \rightarrow +\infty} f(x) $$
4
5 $$ \lim_{x \rightarrow -\infty} f(x) $$
6
7 $$ \lim_{x \rightarrow c} f(x) $$
8
9 Производная (нотация Лагранжа):
10
11 $$ f'(x) $$
12
13 Частная производная (нотация Лейбница):
14
15 $$ \frac{\partial f}{\partial x} $$

```



```

16
17   Градиент:
18
19   $$
20   \nabla f(x_1, x_2) =
21   \begin{bmatrix}
22   \frac{\partial f}{\partial x_1} \quad \frac{\partial f}{\partial x_2}
23   \end{bmatrix}
24   $$
25
26   Интеграл:
27
28   $$\int_a^b f(x)dx$$

```

Пределы:

$$\lim_{x \rightarrow +\infty} f(x)$$

$$\lim_{x \rightarrow -\infty} f(x)$$

$$\lim_{x \rightarrow c} f(x)$$

Производная (нотация Лагранжа):

$$f'(x)$$

Частная производная (нотация Лейбница):

$$\frac{\partial f}{\partial x}$$

Градиент:

$$\nabla f(x_1, x_2) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

Интеграл:

$$\int_a^b f(x)dx$$

Сумма и произведение

```

1   Сумма:
2
3   $$ \sum_{i=1}^n a_{i} $$
4
5   $$\sum_{i=1}^n a_{i} $$
6

```

```

7 Произведение:
8
9 
$$\prod_{j=1}^m a_j$$


```

Сумма:

$$\sum_{i=1}^n a_i$$

$$\sum_{i=1}^n a_i$$

Произведение:

$$\prod_{j=1}^m a_j$$

Матрица

```

1 Без скобок (plain):
2
3 $$
4 \begin{matrix}
5 1 & 2 & 3 \\
6 a & b & c
7 \end{matrix}
8 $$
9
10 Круглые скобки (parentheses, round brackets):
11
12 $$
13 \begin{pmatrix}
14 1 & 2 & 3 \\
15 a & b & c
16 \end{pmatrix}
17 $$
18
19 Квадратные скобки (square brackets):
20
21 $$
22 \begin{bmatrix}
23 1 & 2 & 3 \\
24 a & b & c
25 \end{bmatrix}
26 $$
27
28 Фигурные скобки (curly brackets, braces):

```

```
29  
30 $$  
31 \begin{Bmatrix}  
32 1 & 2 & 3\\  
33 a & b & c  
34 \end{Bmatrix}  
35 $$
```

```
36  
37 Прямые скобки (pipes):
```

```
38  
39 $$  
40 \begin{vmatrix}  
41 1 & 2 & 3\\  
42 a & b & c  
43 \end{vmatrix}  
44 $$
```

```
45  
46 Двойные прямые скобки (double pipes):
```

```
47  
48 $$  
49 \begin{Vmatrix}  
50 1 & 2 & 3\\  
51 a & b & c  
52 \end{Vmatrix}  
53 $$
```

Без скобок (plain):

$$\begin{matrix} 1 & 2 & 3 \\ a & b & c \end{matrix}$$

Круглые скобки (parentheses, round brackets):

$$\begin{pmatrix} 1 & 2 & 3 \\ a & b & c \end{pmatrix}$$

Квадратные скобки (square brackets):

$$\begin{bmatrix} 1 & 2 & 3 \\ a & b & c \end{bmatrix}$$

Фигурные скобки (curly brackets, braces):

$$\begin{Bmatrix} 1 & 2 & 3 \\ a & b & c \end{Bmatrix}$$

Прямые скобки (pipes):

$$\begin{vmatrix} 1 & 2 & 3 \\ a & b & c \end{vmatrix}$$

Двойные прямые скобки (double pipes):

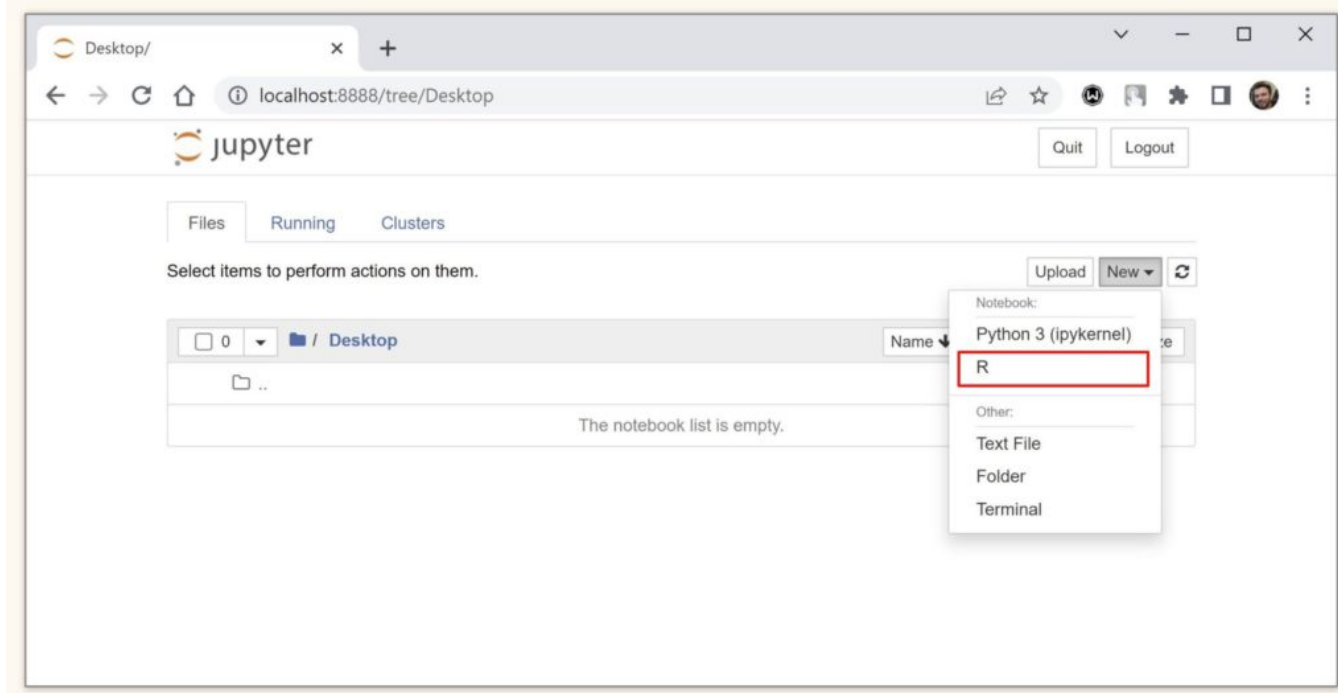
$$\begin{Vmatrix} 1 & 2 & 3 \\ a & b & c \end{Vmatrix}$$

Программирование на R

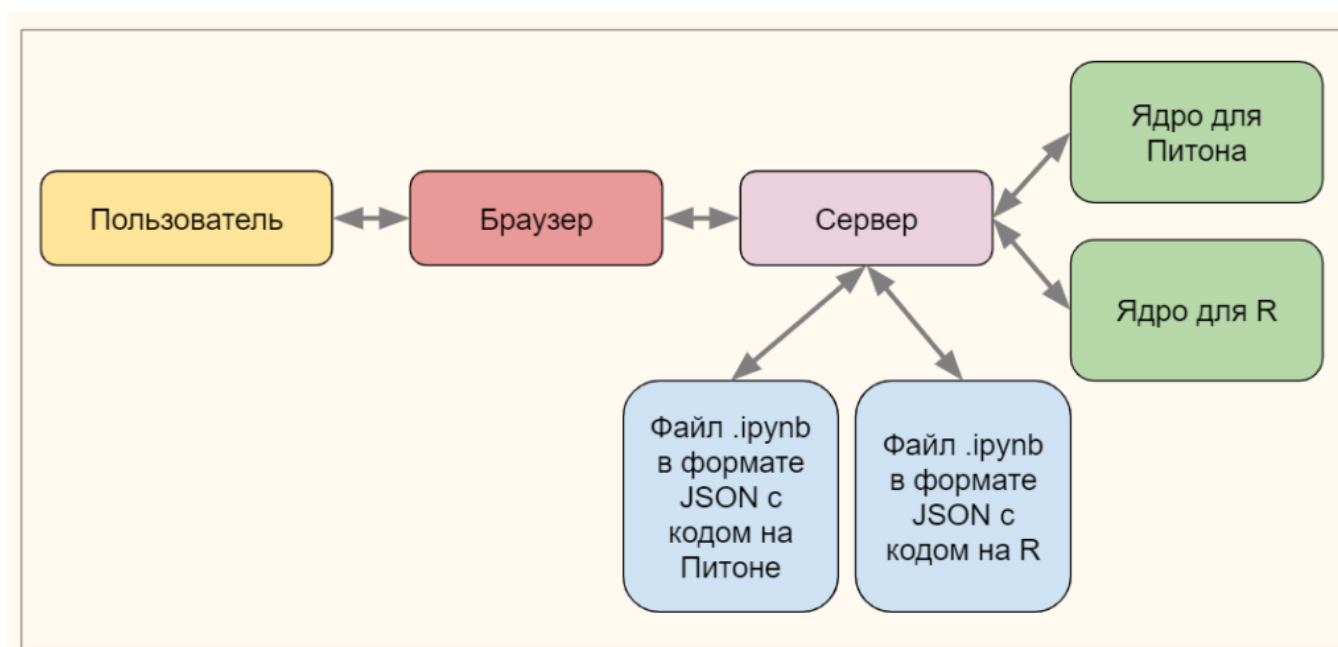
Jupyter Notebook позволяет писать код на других языках программирования, не только на Питоне. Попробуем написать и исполнить код на R, языке, который специально разрабатывался для data science.

Вначале нам понадобится установить kernel для R. Откроем Anaconda Prompt и введем следующую команду `conda install -c r r-irkernel`. В процессе установки система спросит продолжать или нет (Proceed ([y]/n)?). Нажмите **y + Enter**.

Откройте Jupyter Notebook. В списке файлов создайте ноутбук на R. Назовем его **rprogramming**.



После установки нового ядра и создания еще одного файла .ipynb схема работы нашего Jupyter Notebook немного изменилась.



Теперь мы готовы писать код на R. Мы уже начали знакомиться с этим языком, когда изучали парадигмы программирования. Сегодня мы рассмотрим основные типы данных и особенности синтаксиса.

Переменные в R

Числовые, строковые и логические переменные

Как и в Питоне, в R мы можем создавать числовые (numeric), строковые (character) и логические (logical) переменные.

```
1 # поместим число 42 в переменную numeric_var
2 numeric_var = 42
3
4 # строку поместим в переменную text_var
5 text_var <- 'Hello world!'
6
7 # наконец присвоим значение TRUE переменной logical_var
8 TRUE -> logical_var
```

Для присвоения значений можно использовать как оператор `=`, так и операторы присваивания `<-` и `->`. Обратите внимание, используя `->` мы можем поместить значение слева, а переменную справа.

Посмотрим на результат (в Jupyter Notebook можно обойтись без функции `print()`).

```
1 text_var
```

```
1 'Hello world!'
```

Выведем класс созданных нами объектов с помощью функции `class()`.

```
1 class(numeric_var)
2 class(text_var)
3 class(logical_var)
```

```
1 'numeric'
2 'character'
3 'logical'
```

Тип данных можно посмотреть с помощью функции `typeof()`.

```
1 typeof(numeric_var)
2 typeof(text_var)
3 typeof(logical_var)
```

```
1 'double'
2 'character'
3 'logical'
```

Хотя вывод этих функций очень похож, мы, тем не менее, видим, что классу numeric соответствует тип данных double (число с плавающей точкой с двумя знаками после запятой).

Числовые переменные: numeric, double, integer

По умолчанию, в R и целые числа, и дроби хранятся в формате double.

```
1 # еще раз поместим число 42 в переменную numeric_var
2 numeric_var <- 42
3
4 # выведем тип данных
5 typeof(numeric_var)
```

```
1 'double'
```

Принудительно перевести 42 в целочисленное значение можно с помощью функции **as.integer()**.

```
1 int_var <- as.integer(numeric_var)
2 typeof(int_var)
```

```
1 'integer'
```

Кроме того, если после числа поставить L, это число автоматически превратится в integer.

```
1 typeof(42L)
```

```
1 'integer'
```

Превратить integer обратно в double можно с помощью функций **as.double()** и **as.numeric()**.

```
1 typeof(as.double(int_var))
2 typeof(as.numeric(42L))
```

```
1 'double'
2 'double'
```

Если число хранится в формате строки, его можно перевести обратно в число (integer или double).

```
1 text_var <- '42'
2 typeof(text_var)
```

```
1 'character'
```

```
1 typeof(as.numeric(text_var)) # можно также использовать as.double()
2 typeof(as.integer(text_var))
```

```
1 'double'
2 'integer'
```

Вектор

Вектор (vector) — это одномерная структура, которая может содержать множество элементов одного типа. Вектор можно создать с помощью **функции** `c()`.

```
1 # создадим вектор с информацией о продажах товара в магазине за неделю (с)
2 sales <- c(24, 28, 32, 25, 30, 31, 29)
3 sales
```

```
1 24 28 32 25 30 31 29
```

С помощью функций **`length()`** и **`typeof()`** мы можем посмотреть соответственно общее количество элементов и тип данных каждого из них.

```
1 # посмотрим на общее количество элементов и тип данных каждого из них
2 length(sales)
3 typeof(sales)
```

```
1 7
2 'double'
```

У вектора есть **индекс**, который (в отличие, например, от списков в Питоне), начинается с *единицы*.

```
1 sales[1]
```

```
1 24
```

При указании **диапазона** выводятся и первый, и последний его элементы.

```
1 sales[1:5]
```

```
1 24 28 32 25 30
```

Отрицательный индекс убирает элементы из вектора.

```
1 sales[-5]
```

```
1 24 28 32 25 31 29
```

Именованный вектор (named vector) создается с помощью **функции** `names()`.

```
1 # создадим еще один вектор с названиями дней недели
2 days_vector <- c('Понедельник', 'Вторник', 'Среда', 'Четверг', 'Пятница',
3 days_vector
```

```
1 'Понедельник' 'Вторник' 'Среда' 'Четверг' 'Пятница' 'Суббота' 'Воскресенье'
```



```
1 # создадим именованный вектор с помощью функции names()  
2 names(sales) <- days_vector  
3 sales
```

Понедельник	24
Вторник	28
Среда	32
Четверг	25
Пятница	30
Суббота	31
Воскресенье	29

Выводить элементы именованного вектора можно не только по числовому индексу, но и по их названиям.

```
1 sales['Воскресенье']
```

```
1 Воскресенье: 29
```

Список

В отличие от вектора, **список** (list) может содержать множество элементов различных типов.

```
1 # список создается с помощью функции list()  
2 list('DS', 'ML', c(21, 24), c(TRUE, FALSE), 42.0)
```

```
1 [[1]]  
2 [1] "DS"  
3  
4 [[2]]  
5 [1] "ML"  
6  
7 [[3]]  
8 [1] 21 24  
9  
10 [[4]]  
11 [1] TRUE FALSE  
12  
13 [[5]]  
14 [1] 42
```

Матрица

Матрица (`matrix`) в R — это двумерная структура, содержащая одинаковый тип данных (чаще всего числовой). Матрица создается с помощью **функции `matrix()`** с параметрами `data`, `nrow`, `ncol` и `byrow`.

- `data` — данные для создания матрицы
- `nrow` и `ncol` — количество строк и столбцов
- `byrow` — параметр, указывающий заполнять ли элементы матрицы построчно (`TRUE`) или по столбцам (`FALSE`)

Рассмотрим несколько примеров. Создадим последовательность целых чисел (по сути, тоже вектор).

```
1 # для этого подойдет функция seq()
2 sqn <- seq(1:9)
3
4 sqn
5 typeof(sqn)
```

```
1 1 2 3 4 5 6 7 8 9
2 'integer'
```

Используем эту последовательность для создания двух матриц.

```
1 # создадим матрицу, заполняя значения построчно
2 mtx <- matrix(sqn, nrow = 3, ncol = 3, byrow = TRUE)
3 mtx
```

```
1 2 3
```

```
4 5 6
```

```
7 8 9
```

```
1 # теперь создадим матрицу, заполняя значения по столбцам
2 mtx <- matrix(sqn, nrow = 3, ncol = 3, byrow = FALSE)
3 mtx
```

1 4 7

2 5 8

3 6 9

Зададим названия для строк и столбцов второй матрицы.

```
1 # создадим два вектора с названиями строк и столбцов
2 rows <- c('Row 1', 'Row 2', 'Row 3')
3 cols <- c('Col 1', 'Col 2', 'Col 3')
4
5 # используем функции rownames() и colnames(),
6 # чтобы передать эти названия нашей матрице
7 rownames(mtx) <- rows
8 colnames(mtx) <- cols
9
10 # посмотрим на результат
11 mtx
```

Посмотрим на размерность этой матрицы с помощью **функции dim()**.

```
1 dim(mtx)
```

```
1 3 3
```

Массив

В отличие от матрицы, **массив** (array) — это многомерная структура. Создадим трехмерный массив размерностью 3 x 2 x 3. Вначале создадим три матрицы размером 3 x 2.

```
1 # создадим три матрицы размером 3 x 2,
2 # заполненные пятерками, шестерками и семерками
3 a <- matrix(5, 3, 2)
4 b <- matrix(6, 3, 2)
```

```
5 | c <- matrix(7, 3, 2)
```

Теперь соединим их с помощью **функции array()**. Передадим этой функции два параметра в форме векторов: данные (data) и размерность (dim).

```
1 | arr <- array(data = c(a, b, c), # вектор с матрицами
2 |             dim = c(3, 2, 3)) # вектор размерности
3 |
4 | print(arr)
```

```
1 | , , 1
2 |
3 |      [,1] [,2]
4 | [1,]    5    5
5 | [2,]    5    5
6 | [3,]    5    5
7 |
8 | , , 2
9 |
10 |     [,1] [,2]
11 | [1,]    6    6
12 | [2,]    6    6
13 | [3,]    6    6
14 |
15 | , , 3
16 |
17 |     [,1] [,2]
18 | [1,]    7    7
19 | [2,]    7    7
20 | [3,]    7    7
```

Факторная переменная

Факторная переменная или **фактор** (factor) — специальная структура для хранения категориальных данных. Вначале немного теории.

Как мы узнаем на курсе анализа данных, категориальные данные бывают номинальными и порядковыми. *Номинальные категориальные* (nominal categorical) данные представлены категориями, в которых нет естественного внутреннего порядка. Например, пол или цвет волос человека, марка автомобиля могут быть отнесены к определенным категориям, но не могут быть упорядочены.

Порядковые категориальные (ordinal categorical) данные наоборот обладают внутренним, свойственным им порядком. К таким данным относятся шкала удовлетворенности потребителей, класс железнодорожного билета, должность или звание, а также любая количественная переменная, разбитая на категории (например, низкий, средний и высокий уровень зарплат).

Посмотрим, как учесть такие данные с помощью R. Начнем с номинальных данных.

```
1 # предположим, что мы собрали данные о цветах нескольких автомобилей и по
2 color_vector <- c('blue', 'blue', 'white', 'black', 'yellow', 'white', 'v
3
4 # преобразуем этот вектор в фактор с помощью функции factor()
5 factor_color <- factor(color_vector)
6 factor_color
```

Как вы видите, **функция factor()** разбила данные на категории, при этом эти категории остались неупорядоченными. Посмотрим на класс созданного объекта.

```
1 class(factor_color)
```

```
1 'factor'
```

Теперь поработаем с порядковыми данными.

```
1 # возьмем данные измерений температуры, выраженные категориями
2 temperature_vector <- c('High', 'Low', 'High', 'Low', 'Medium', 'High',
3
4 # создадим фактор
5 factor_temperature <- factor(temperature_vector,
6                               # указав параметр order = TRUE
7                               order = TRUE,
8                               # а также вектор упорядоченных категорий
9                               levels = c('Low', 'Medium', 'High'))
10
11 # посмотрим на результат
12 factor_temperature
```

Выведем класс созданного объекта.

```
1 class(factor_temperature)
```

```
1 'ordered' 'factor'
```

Добавлю, что количество элементов в каждой из категорий можно посмотреть с помощью **функции summary()**.

```
1 summary(factor_temperature)
```

Датафрейм

Датафрейм в R выполняет примерно ту же функцию, что и в Питоне. С помощью **функции data.frame()** создадим простой датафрейм, где параметрами будут названия столбцов, а аргументами — векторы их значений.

```
1 df <- data.frame(city = c('Москва', 'Париж', 'Лондон'),  
2                   population = c(12.7, 2.1, 8.9),  
3                   country = c('Россия', 'Франция', 'Великобритания'))  
4  
5 df
```

Доступ к элементам датафрейма можно получить по **индексам строк и столбцов**, которые также начинаются с *единицы*.

```
1 # выведем значения первой строки и первого столбца
2 df[1, 1]
```

```
1 # выведем всю первую строку
2 df[1, ]
```

```
1 # выведем второй столбец
2 df[, 2]
```

```
1 12.7 2.1 8.9
```

Получить доступ к столбцам можно и так.

```
1 df$population
```

```
1 12.7 2.1 8.9
```

Дополнительные пакеты

Как и в Питоне, в R мы можем установить дополнительные пакеты через Anaconda Prompt. Например, установим пакет `ggplot2` для визуализации данных. Для этого введем команду `conda install r-ggplot2`.

В целом команда установки пакетов для R следующая: `conda install r-<package_name>`.

Продemonстрируем работу с этим пакетом с помощью несложного датасета `mtcars`.

```
1 # импортируем библиотеку datasets
2 library(datasets)
3
```

```
4 # загрузим датасет mtcars
5 data(mtcars)
6
7 # выведем его на экран
8 mtcars
```

Примечание. Здесь приведена лишь часть датасета.

Теперь импортируем установленную ранее библиотеку ggplot2.

```
1 library(ggplot2)
```

Построим гистограмму по столбцу mpg (miles per galon, расход в милях на галлон топлива). Для построения гистограммы нам потребуется через «+» объединить две функции:

- **функцию ggplot()**, которой мы передадим наши данные и еще одну *функцию aes()*, от англ. aesthetics, которая свяжет ось x нашего графика и столбец данных mpg, а также

- функцию **geom_histogram()** с параметрами `bins` (количество интервалов) и `binwidth` (их ширина), которая и будет отвечать за создание гистограммы

```
1 # данными будет датасет mtcars, столбцом по оси x - mpg
2 ggplot(data = mtcars, aes(x = mpg)) +
3
4 # типом графика будет гистограмма с 10 интервалами шириной 5 миль на галл
5 geom_histogram(bins = 10, binwidth = 5)
```

Примерно также мы можем построить график плотности распределения (density plot). Только теперь мы передадим функции `aes()` еще один параметр `fill = as.factor(vs)`, который (предварительно превратив столбец в фактор через `as.factor()`) позволит разбить данные на две категории по столбцу `vs`. В этом датасете признак `vs` указывает на конфигурацию двигателя (расположение цилиндров), v-образное, v-shaped (`vs == 0`) или рядное, straight (`vs == 1`).

Кроме того, для непосредственного построения графика мы будем использовать новую функцию **geom_density()** с параметром `alpha`, отвечающим за прозрачность заполнения пространства под кривыми.

```
1 ggplot(data = mtcars, aes(x = mpg, fill = as.factor(vs))) +
```

```
2 | geom_density(alpha = 0.3)
```

Дополнительно замечу, что к столбцам датафрейма можно применять множество различных функций, например, рассчитать среднее арифметическое или медиану с помощью несложных для запоминания `mean()` и `median()`.

```
1 | mean(mtcars$mpg)
2 | median(mtcars$mpg)
```

```
1 | 20.090625
2 | 19.2
```

Кроме того, мы можем применить уже знакомую нам функцию **`summary()`**, которая для количественного столбца выдаст минимальное и максимальное значения, первый (Q1) и третий (Q2) квартили, а также медиану и среднее значение.

```
1 | summary(mtcars$mpg)
```

1	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
2	10.40	15.43	19.20	20.09	22.80	33.90

В файле ниже содержится созданный нами код на R.

rprogramming.zip

Скачать

Вернемся к основной теме занятия.

Подробнее про Anaconda

Conda

Программа conda, как уже было сказано, объединяет в себе систему управления *пакетами* (как pip) и, кроме того, позволяет создавать *окружения*.

Идея **виртуального окружения** (virtual environment) заключается в том, что если в рамках вашего проекта вы, например, используете определенную версию библиотеки Numpy и установка более ранней или более поздней версии приведет к сбоям в работе вашего кода, хорошим решением была бы изоляция нужной версии Numpy, а также всех остальных используемых вами библиотек. Именно для этого и нужно виртуальное окружение.

Рассмотрим, как мы можем устанавливать пакеты и создавать окружения через Anaconda Prompt и через Anaconda Navigator.

Anaconda Prompt

Про пакеты. По аналогии с pip, установленные (в текущем окружении) пакеты можно посмотреть с помощью команды `conda list`.

Установить пакет можно с помощью команды `conda install <package_name>`. Обновить пакет можно через `conda update <package_name>`. Например, снова попробуем установить Numpy. o

Про окружения. По умолчанию мы работаем в базовом окружении (base environment). Посмотреть, какие в целом установлены окружения можно с помощью команды `conda info --envs`.

Как вы видите, пока у нас есть только одно окружение. Давайте создадим еще одно виртуальное окружение и назовем его, например, waterfall.

Введите команду `conda create --name waterfall`.

Введем две команды

- `conda activate waterfall` для активации нового окружения
- `conda list` для того, чтобы посмотреть установленные в нем пакеты

Как вы видите, в новом окружении нет ни одного пакета. Введем `conda search seaborn`, чтобы посмотреть какие версии этого пакета доступны для скачивания.

Скачаем этот пакет через `conda install seaborn`. Проверим установку с помощью `conda list`.

Как вы видите, помимо seaborn было установлено множество других необходимых для работы пакета библиотек. Вернуться в базовое окружение

можно с помощью команд `conda activate base` или `conda deactivate`.

Импорт модулей и переменная path

На прошлом занятии мы научились импортировать собственный модуль в командной строке Windows (cmd).

Посмотрим, отличается ли содержимое списка path для двух установленных версий Питона. Для этого в командной строке Windows и в Anaconda Prompt перейдем в интерактивный режим с помощью `python`. Затем введем

```
1 import sys
2 sys.path
```

Как мы видим, пути в переменной path будут отличаться и это нужно учитывать, если мы хотим локально запускать собственные модули.

Anaconda Navigator

Запускать программы, управлять окружениями и устанавливать необходимые библиотеки можно также через Anaconda Navigator. На вкладке **Home** вы видите программы, которые можно *открыть* (launch) или *установить* (install) для текущего окружения.

На вкладке **Environments** отображаются созданные нами окружения (в частности, окружение `waterfall`, которое мы создали ранее) и содержащиеся в них пакеты.

В целом интерфейс интуитивно понятен, и так как мы уже познакомились с принципом создания окружений и установки в них дополнительных пакетов, уверен, работа с Anaconda Navigator сложностей не вызовет.

Прежде чем завершить, обратимся к еще одной программе для интерактивного программирования JupyterLab.

JupyterLab

JupyterLab — расширенная версия Jupyter Notebook, которая также входит в дистрибутив Anaconda. Запустить эту программу можно через Anaconda Navigator или введя команду `jupyter lab` в Anaconda Prompt.

После запуска вы увидите вкладку **Launcher**, в которой можно создать новый *ноутбук* (Notebook) на Питоне или R, открыть *консоль* (Console) на этих языках, а также создать файлы в различных форматах (Other). Слева вы видите список папок компьютера.

В разделе Console нажмем на Python 3 (ipykernel). Введем несложный код (см. ниже) и исполним его, нажимая **Shift + Enter**.

Как вы видите, здесь мы можем писать код на Питоне так же, как мы это делали в командной строке Windows на прошлом занятии. Закроем консоль.

В файловой системе слева мы можем открывать уже созданные ноутбуки. Например, откроем ноутбук на R **rprogramming.ipynb**.

В левом меню на второй сверху вкладке мы видим открытые горизонтальные вкладки (Launcher и rprogramming.ipynb), а также запущенные ядра (kernels).

Консольные ядра (Console 1 и Console 2) можно открыть (по сути, мы снова запустим консоль).

Две оставшиеся вертикальные вкладки открывают доступ к автоматическому оглавлению (content) и расширениям (extensions).

Вкладки Run и Kernel в верхнем меню JupyterLab в целом аналогичны вкладкам Cell и Kernel в JupyterNotebook.

Подведем итог

На сегодняшнем занятии мы познакомились с программой Jupyter Notebook, а также изучили дистрибутив Anaconda, в состав которого входит эта программа.

Говоря о **программе Jupyter Notebook**, мы узнали про возможности работы с ячейками и ядром программы. Кроме того, мы познакомились с языком разметки Markdown и написанием формул с помощью языка верстки LaTeX.

После этого мы установили ядро для программирования на R и рассмотрели основы этого языка.

При изучении **дистрибутива Anaconda** мы познакомились с системой conda и попрактиковались в установке библиотек и создании окружений через Anaconda Prompt и Anaconda Navigator.

Наконец мы узнали про особенности программы JupyterLab.

Вопросы для закрепления

Вопрос. Что такое Anaconda?

Посмотреть правильный ответ

Вопрос. Какой тип ячеек доступен в Jupyter Notebook?

Посмотреть правильный ответ

Вопрос. Для чего нужно виртуальное окружение?

Посмотреть правильный ответ

Ответы на вопросы

Вопрос. Можно ли исполнить код на R в Google Colab?

Ответ. Да, это возможно. Причем двумя способами.

Способ 1. Откройте ноутбук. Введите и выполните команду `%load_ext rpy2.ipython`. В последующих ячейках введите `%R`, чтобы в этой же строке написать код на R или `%%R`, если хотите, чтобы вся ячейка исполнилась как код на R (так называемые магические команды).

В этом случае мы можем исполнять код на двух языках внутри одного ноутбука.

```
1 # введем магическую команду, которая позволит программировать на R
2 %load_ext rpy2.ipython
```

```
1 # команда %%R позволит Colab распознать ячейку как код на R
2 %%R
3
4 # кстати, числовой вектор можно создать просто с помощью двоеточия
5 x <- 1:10
6 x
```

```
1 [1] 1 2 3 4 5 6 7 8 9 10
```

```
1 # при этом ничто не мешает нам продолжать писать код на Питоне
2 import numpy as np
3 np.mean([1, 2, 3])
```

```
1 2.0
```

Приведенный выше код можно найти в [дополнительных материалах](#) к занятию.

Способ 2. Если вы хотите, чтобы весь код исполнялся на R (как мы это делали в Jupyter Notebook), создайте новый ноутбук используя одну из ссылок ниже:

- <https://colab.research.google.com/#create=true&language=r>
- <https://colab.to/r>

Теперь, если вы зайдете на вкладку **Runtime** → **Change runtime type**, то увидите, что можете выбирать между Python и R.

Выведем версию R в Google Colab.

```
1 | R.version.string
```

```
1 | 'R version 4.2.0 (2022-04-22)'
```

Посмотреть на установленные пакеты можно с помощью `installed.packages()`.
Созданный ноутбук Google Colab на R доступен [по ссылке](#).

Вопрос. Очень медленно загружается Anaconda. Можно ли что-то сделать?

Ответ. Можно работать через Anaconda Prompt, эта программа быстрее графического интерфейса Anaconda Navigator.

Кроме того, можно использовать дистрибутив Miniconda, в который входит conda, Питон и несколько ключевых пакетов. Остальные пакеты устанавливаются вручную по мере необходимости.

Вопрос. Разве Jupyter не должен писаться через i, как Jupiter?

Ответ. Вы правы в том плане, что название Jupyter Notebook происходит не от планеты Юпитер, которая по-английски как раз пишется через i (Jupiter), а

представляет собой акроним от названий языков программирования Julia, Python и R.

При этом, как утверждают разработчики^[1], слово Jupyter также отсылает к тетрадам (notebooks) Галилея, в которых он, в частности, документировал наблюдение за лунами Юпитера.

Вопрос. В каких еще программах можно писать код на Питоне и R?

Ответ. Таких программ несколько. Довольно удобно пользоваться облачным решением Kaggle. Там можно создавать как скрипты (scripts, в том числе RMarkdown Scripts), так и ноутбуки на Питоне и R. Подробнее можно почитать в документации^[2] на их сайте.

Вопрос. Можно ли создать виртуальное окружение каким-либо другим способом помимо программы conda?

Ответ. Да, можно. Вот здесь^[3] есть хорошая видео-инструкция.

Вот коротко какие шаги нужно выполнить.

Вначале убедитесь, что у вас уже установлен Питон. В нем по умолчанию содержится **модуль venv**, который как раз предназначен для создания виртуального окружения.

Шаг 1. Создайте папку с вашим проектом, например, пусть это будет папка *webapp* для веб-приложения на популярном фреймворке для Питона Django.

Шаг 2. В командной строке перейдите в папку webapp.

Затем введите команду для создания виртуального окружения.

```
1 | python -m venv djenv
```

По сути мы говорим Питону создать окружение djenv (название может быть любым) с помощью модуля venv. Переключатель (flag или switch) `-m` подсказывает питону, что venv — это модуль, а не файл.

После выполнения этой команды создается папка djenv виртуального окружения.

Шаг 3. Активируем это виртуальное окружение следующей командой.

```
1 | .\djenv\Scripts\activate
```

Здесь мы обращаемся к файлу `activate` внутри папки *Scripts*. Как вы видите, название окружения появилось слева от пути к папке.

Теперь через `pip` можно устанавливать пакеты, которые будут «видны» только внутри виртуального окружения `djenv`.

Шаг 4. Выйти из этого виртуального окружения можно с помощью команды `deactivate`. Если вам нужно удалить окружение, сначала деактивируйте его, а затем вручную удалите соответствующую папку.

На следующем занятии мы поговорим про такую важную тему, как регулярные выражения.