

# Введение

С.В. Лемешевский ([sergey.lemeshevsky@gmail.com](mailto:sergey.lemeshevsky@gmail.com))

Институт математики НАН Беларуси

Feb 22, 2020

## Содержание

<b>1 Установка</b>	<b>1</b>
1.1 Версии Python . . . . .	1
1.2 Установка Python . . . . .	2
1.3 Установка Anaconda . . . . .	5
1.4 Проверка работоспособности . . . . .	11
<b>2 Установка библиотек</b>	<b>14</b>
2.1 Способы обновления библиотек . . . . .	15
<b>3 Среды для вычислений в Python</b>	<b>16</b>
3.1 Интерпретатор . . . . .	16
3.2 Консоль IPython . . . . .	17
3.3 Интерпретатор и текстовый редактор как среда разработки . . . . .	24
3.4 Jupyter . . . . .	25
3.5 Spyder: Интегрированная среда разработки . . . . .	33

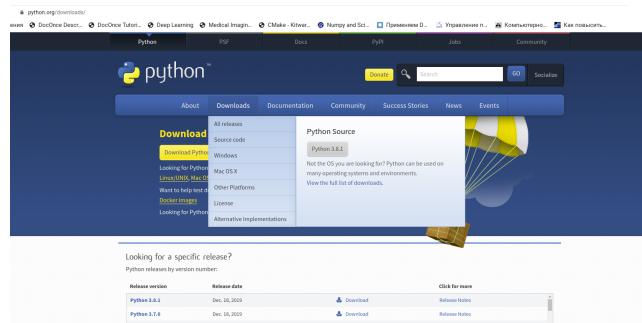
## 1. Установка

### 1.1. Версии Python

На сегодняшний день существуют две версии Python — это Python 2 и Python 3, у них отсутствует полная совместимость друг с другом. На данный момент вторая версия Python ещё широко используется, но, судя по изменениям, которые происходят, со временем, она останется только для запуска старого кода. Мы будем Python 3, и, в дальнейшем, если где-то будет встречаться слово Python, то под ним следует понимать Python 3. Случай применения Python 2 будут специально оговариваться.

## 1.2. Установка Python

Для установки интерпретатора Python на ваш компьютер, первое, что нужно сделать — это скачать дистрибутив. Загрузить его можно с официального сайта, перейдя по ссылке <https://www.python.org/downloads/>



**Установка Python в Windows.** Для операционной системы Windows дистрибутив распространяется либо в виде исполняемого файла, либо в виде архивного файла.

Files						
Version	Operating System	Description	MD5 Sum	File Size	GPG	
Gzipped source tarball	Source release	f215a975a57b6e789c1787ec56b2bd	23978360	SIG		
XZ compressed source tarball	Source release	b3fb65547f070bf950c626fe9ca6307	17638408	SIG		
macOS 64-bit installer	Mac OS X	for OS X 10.9 and later	d1b96953126b614e11b036a45103	29051411	SIG	
Windows help file	Windows	f0bb04c39f1de380f61625e6a6f2	8480993	SIG		
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64	409f387fa215d940060552b211061	8013540	SIG	
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64	3e4c42f5ff8bebe828c9127a7eb1	27543360	SIG	
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64	602961723cc54783973302789d6145	1363800	SIG	
Windows x86 embeddable zip file	Windows		98063745a7e5259e5ab40443a07734	7143308	SIG	
Windows x86 executable installer	Windows		2d4c7def7d6fc8231c3dec8d1abf79	26446128	SIG	
Windows x86 web-based installer	Windows		d11706da544e7a968e32bb0520f51	1325432	SIG	

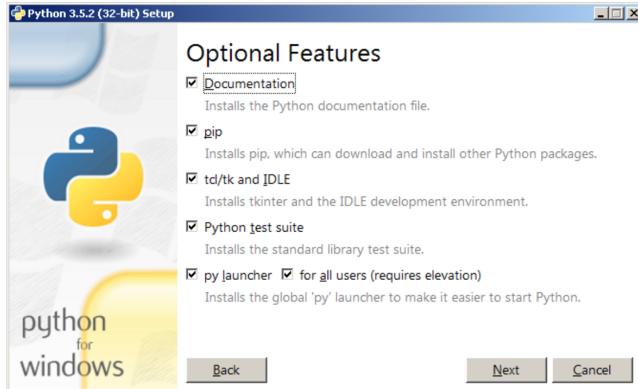
### Порядок установки.

1. Запустите скачанный установочный файл.
2. Выберете способ установки.



В данном окне предлагается два варианта *Install Now* и *Customize installation*. При выборе *Install Now*, Python установится в папку по указанному пути. Помимо самого интерпретатора будет установлен **IDLE** (интегрированная среда разработки), **pip** (пакетный менеджер) и документация, а также будут созданы соответствующие ярлыки и установлены связи файлов, имеющие расширение .ру с интерпретатором Python. *Customize installation* – это вариант настраиваемой установки. Опция *Add python 3. to PATH* нужна для того, чтобы появилась возможность запускать интерпретатор без указания полного пути до исполняемого файла при работе в командной строке.

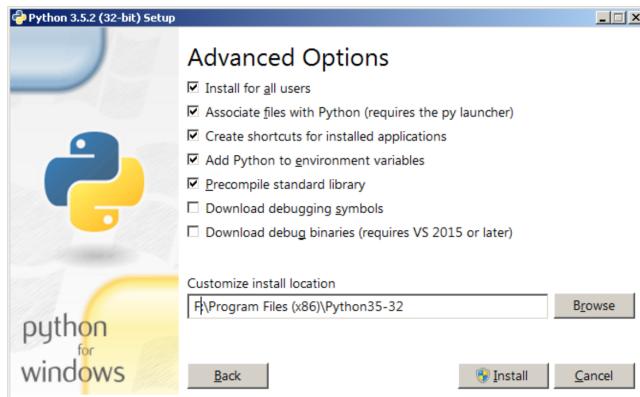
1. Отметьте необходимые опции установки (доступно при выборе *Customize installation*)



На этом шаге нам предлагается отметить дополнения, устанавливаемые вместе с интерпретатором Python . Рекомендуем выбрать все опции:

- **Documentation** – установка документаций.

- **pip** – установка пакетного менеджера pip.
- **tcl/tk and IDLE** – установка интегрированной среды разработки (IDLE) и библиотеки для построения графического интерфейса (tkinter).
- Выберете место установки (доступно при выборе Customize installation )

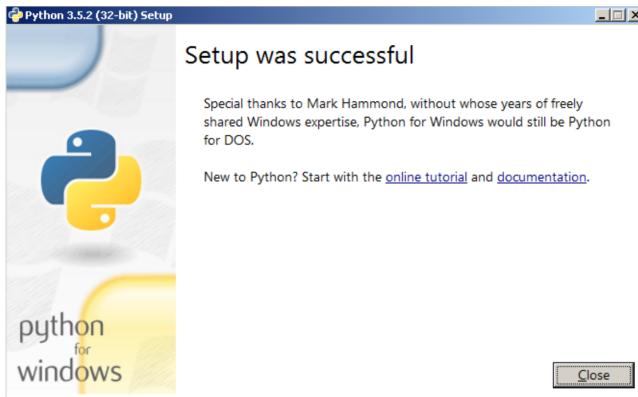


Помимо указания пути, данное окно позволяет внести дополнительные изменения в процесс установки с помощью опций:

- **Install for all users** – Установить для всех пользователей. Если не выбрать данную опцию, то будет предложен вариант инсталляции в папку пользователя, устанавливающего интерпретатор.
- **Associate files with Python** – Связать файлы, имеющие расширение .py, с Python. При выборе данной опции будут внесены изменения в Windows, позволяющие запускать Python скрипты по двойному щелчку мыши.
- **Create shortcuts for installed applications** – Создать ярлыки для запуска приложений.
- **Add Python to environment variables** – Добавить пути до интерпретатора Python в переменную PATH.
- **Precompile standard library** – Провести прекомпиляцию стандартной библиотеки.

Последние два пункта (Download debugging symbols, Download debug binaries) связаны с загрузкой компонентов для отладки, их мы устанавливать не будем.

1. После успешной установки вас ждет следующее сообщение.



**Установка Python в Linux.** Чаще всего интерпретатор Python уже входит в состав дистрибутива. Это можно проверить набрав в терминале

---

```
> python
```

---

ИЛИ

---

```
> python3
```

---

В первом случае, вы запустите Python 2 во втором – Python 3. В будущем, скорее всего, во всех дистрибутивах Linux, включающих Python, будет входить только третья версия. Если у вас, при попытке запустить Python, выдается сообщение о том, что он не установлен, или установлен, но не тот, что вы хотите, то у вас есть два пути: а) собрать Python из исходников; б) взять из репозитория.

Например, для установки из репозитория в Ubuntu воспользуйтесь командой:

---

```
> sudo apt-get install python3
```

---

### 1.3. Установка Anaconda

Для удобства запуска примеров и изучения языка Python, советуем установить на свой ПК пакет Anaconda. Этот пакет включает в себя интерпретатор языка Python (есть версии 2 и 3), набор наиболее часто используемых библиотек и удобную среду разработки и исполнения, запускаемую в браузере.

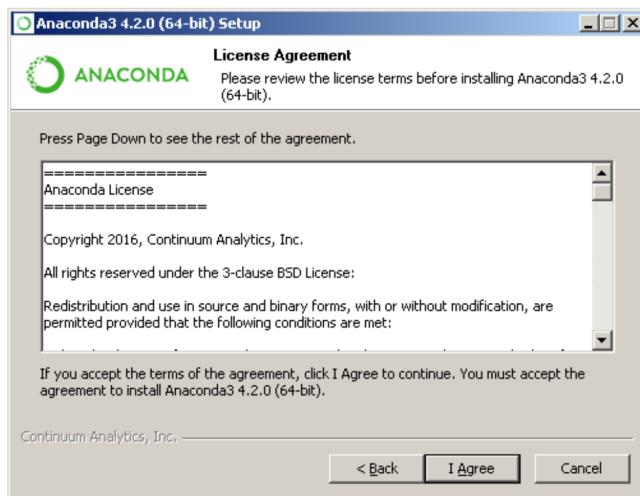
Для установки этого пакета, предварительно нужно скачать дистрибутив <https://www.continuum.io/downloads>.

Есть варианты под Windows, Linux и Mac OS.

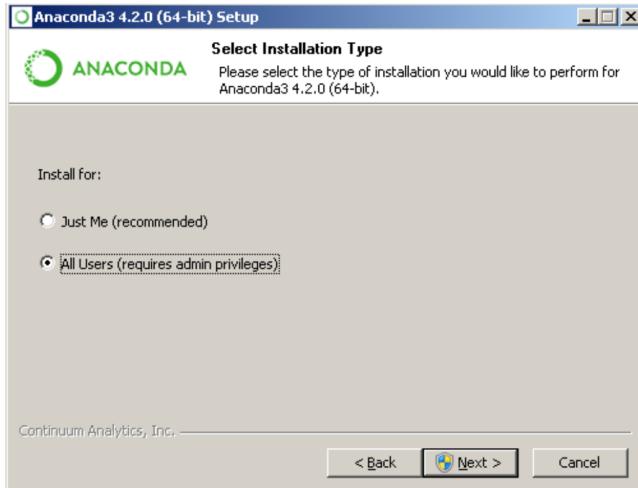
**Установка Anaconda в Windows.** 1. Запустите скачанный инсталлятор. В первом появившемся окне необходимо нажать «Next».



2. Далее следует принять лицензионное соглашение.

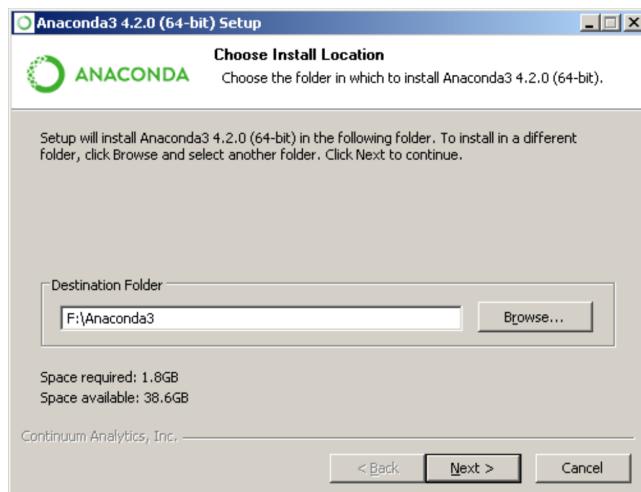


3. Выберете одну из опций установки:

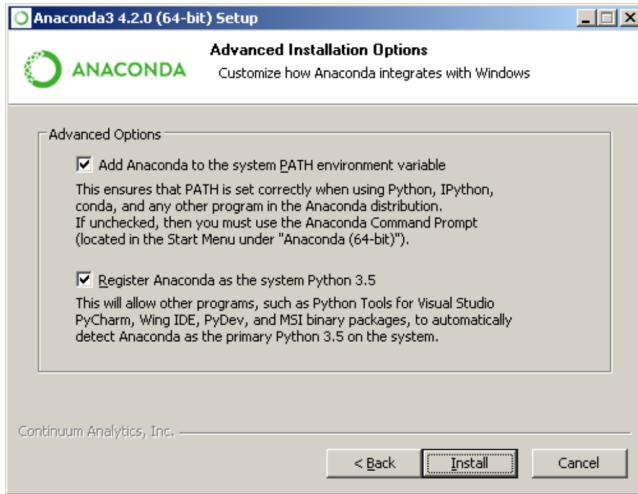


- Just Me – только для пользователя, запустившего установку;
- All Users – для всех пользователей.

4. Укажите путь, по которому будет установлена Anaconda.



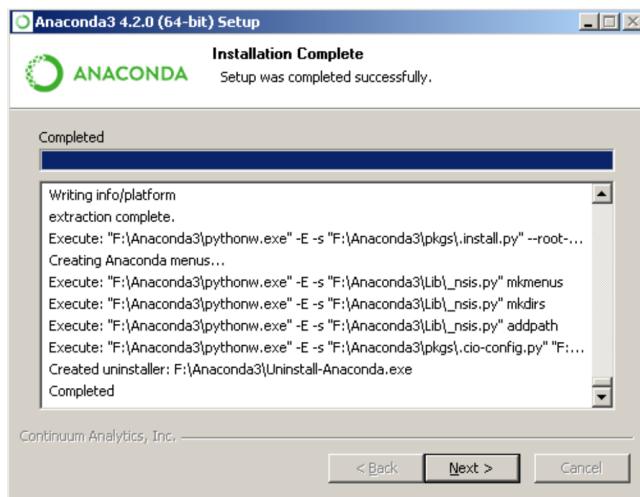
5. Укажите дополнительные опции:



- **Add Anaconda to the system PATH environment variable** – добавить Anaconda в системную переменную PATH;
- **Register Anaconda as the system Python 3** – использовать Anaconda, как интерпретатор Python 3 по умолчанию.

Для начала установки нажмите на кнопку «Install».

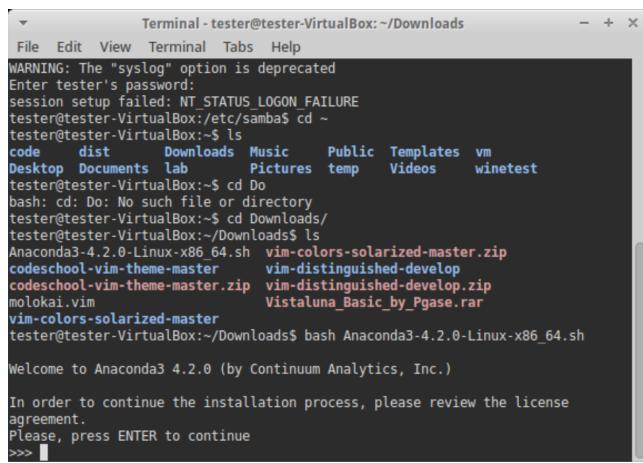
5. После этого будет произведена установка Anaconda на ваш компьютер.



**Установка Anaconda в Linux.** Скачайте дистрибутив Anaconda для Linux, он будет иметь расширение `.sh`, и запустите установку командой:

```
----- [Terminal] -----
> bash имя_дистрибутива.sh
-----
```

В результате вы увидите приглашение к установке. Для продолжения процесса нажмите «Enter».



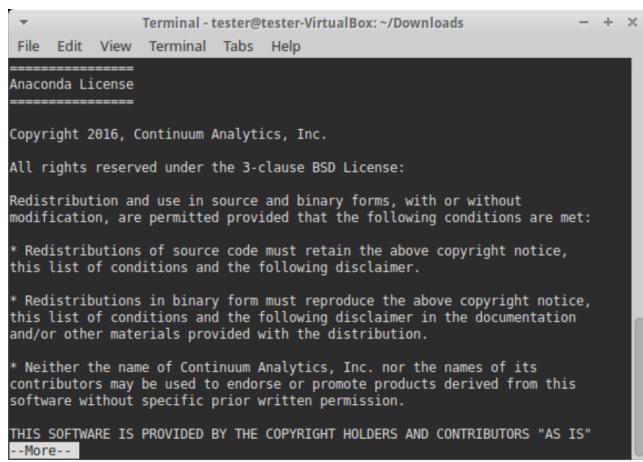
Terminal - tester@tester-VirtualBox: ~/Downloads

```
File Edit View Terminal Tabs Help
WARNING: The "syslog" option is deprecated
Enter tester's password:
session setup failed: NT_STATUS_LOGON_FAILURE
tester@tester-VirtualBox:~/etc/samba$ cd ~
tester@tester-VirtualBox:~$ ls
code dist Downloads Music Public Templates vm
Desktop Documents lab Pictures temp Videos winetest
tester@tester-VirtualBox:~$ cd Do
bash: cd: Do: No such file or directory
tester@tester-VirtualBox:~$ cd Downloads/
tester@tester-VirtualBox:~/Downloads$ ls
Anaconda3-4.2.0-Linux-x86_64.sh vim-colors-solarized-master.zip
codeschool-vim-theme-master vim-distinguished-develop
codeschool-vim-theme-master.zip vim-distinguished-develop.zip
mokai.vim Vistaluna_Basic_by_Pgase.rar
vim-colors-solarized-master
tester@tester-VirtualBox:~/Downloads$ bash Anaconda3-4.2.0-Linux-x86_64.sh

Welcome to Anaconda3 4.2.0 (by Continuum Analytics, Inc.)

In order to continue the installation process, please review the license
agreement.
Please, press ENTER to continue
>>> |
```

2. Прочитайте лицензионное соглашение, его нужно пролистать до конца.



Terminal - tester@tester-VirtualBox: ~/Downloads

```
File Edit View Terminal Tabs Help
=====
Anaconda License
=====

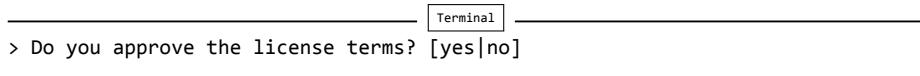
Copyright 2016, Continuum Analytics, Inc.

All rights reserved under the 3-clause BSD License:

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:
* Redistributions of source code must retain the above copyright notice,
  this list of conditions and the following disclaimer.
* Redistributions in binary form must reproduce the above copyright notice,
  this list of conditions and the following disclaimer in the documentation
  and/or other materials provided with the distribution.
* Neither the name of Continuum Analytics, Inc. nor the names of its
  contributors may be used to endorse or promote products derived from this
  software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
--More--|
```

Согласитесь с ним, для этого требуется набрать в командной строке `yes`, в ответе на вопрос инсталлятора:



```
Terminal - tester@tester-VirtualBox:~/Downloads
File Edit View Terminal Tabs Help
cryptography:

openssl
The OpenSSL Project is a collaborative effort to develop a robust,
commercial-grade, full-featured, and Open Source toolkit implementing the
Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols as
well as a full-strength general purpose cryptography library.

pycrypto
A collection of both secure hash functions (such as SHA256 and RIPEMD160),
and various encryption algorithms (AES, DES, RSA, ElGamal, etc.).

pyopenssl
A thin Python wrapper around (a subset of) the OpenSSL library.

kerberos (krb5, non-Windows platforms)
A network authentication protocol designed to provide strong authentication
for client/server applications by using secret-key cryptography.

cryptography
A Python library which exposes cryptographic recipes and primitives.

Do you approve the license terms? [yes|no]
>>> yes
```

3. Выберете место установки. Можно выбрать один из следующих вариантов:

```
Terminal - tester@tester-VirtualBox:~/Downloads
File Edit View Terminal Tabs Help
A collection of both secure hash functions (such as SHA256 and RIPEMD160),
and various encryption algorithms (AES, DES, RSA, ElGamal, etc.).

pyopenssl
A thin Python wrapper around (a subset of) the OpenSSL library.

kerberos (krb5, non-Windows platforms)
A network authentication protocol designed to provide strong authentication
for client/server applications by using secret-key cryptography.

cryptography
A Python library which exposes cryptographic recipes and primitives.

Do you approve the license terms? [yes|no]
>>> yes

Anaconda3 will now be installed into this location:
/home/tester/anaconda3

- Press ENTER to confirm the location
- Press CTRL-C to abort the installation
- Or specify a different location below

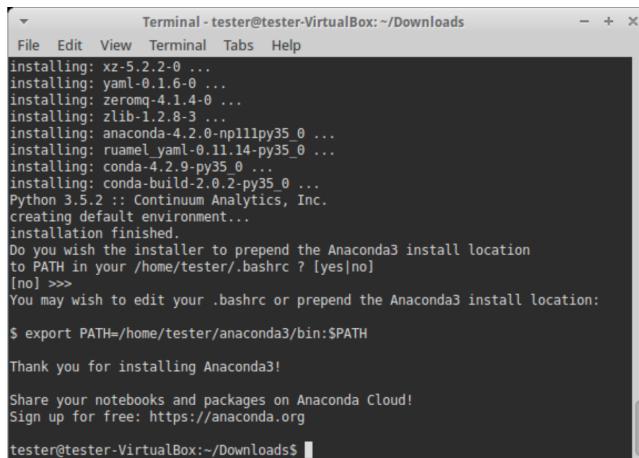
[/home/tester/anaconda3] >>>
```

- **Press ENTER to confirm the location** – нажмите ENTER для принятия предложенного пути установки. Путь по умолчанию для моей машины: /home/tester/anaconda3 , он представлен чуть выше данного меню.
- **Press CTRL-C to abort the installation** – нажмите CTRL-C для отмены установки.

- Or specify a different location below – или укажите другой путь в строке ниже.

Нажмите «ENTER».

4. После этого начнется установка.



The screenshot shows a terminal window titled "Terminal - tester@tester-VirtualBox: ~/Downloads". It displays the following output:

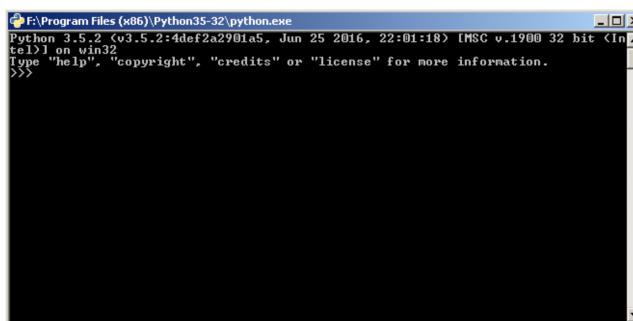
```
File Edit View Terminal Tabs Help
installing: xz-5.2.2-0 ...
installing: yaml-0.1.6-0 ...
installing: zeromq-4.1.4-0 ...
installing: zlib-1.2.8-3 ...
installing: anaconda-4.2.0-np111py35_0 ...
installing: ruamel_yaml-0.11.14-py35_0 ...
installing: conda-4.2.9-py35_0 ...
installing: conda-build-2.0.2-py35_0 ...
Python 3.5.2 :: Continuum Analytics, Inc.
creating default environment...
installation finished.
Do you wish the installer to prepend the Anaconda3 install location
to PATH in your /home/tester/.bashrc ? [yes|no]
[no] >>>
You may wish to edit your .bashrc or prepend the Anaconda3 install location:
$ export PATH=/home/tester/anaconda3/bin:$PATH
Thank you for installing Anaconda3!
Share your notebooks and packages on Anaconda Cloud!
Sign up for free: https://anaconda.org
tester@tester-VirtualBox:~/Downloads$
```

## 1.4. Проверка работоспособности

Теперь проверим работоспособность всего того, что мы установили.

**Проверка интерпретатора Python.** Для начала протестируем интерпретатор в командном режиме. Если вы работаете в Windows , то нажмите сочетание Win+R и в появившемся окне введите python. В Linux откройте окно терминала и в нем введите python3 (или python).

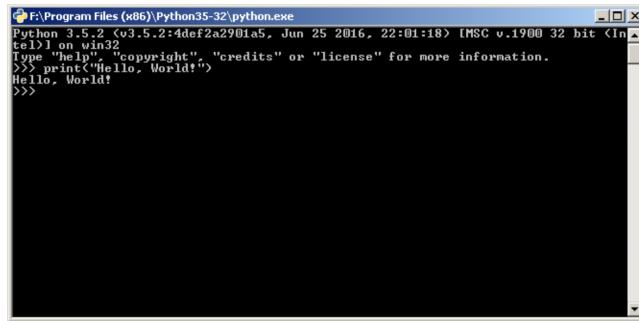
В результате Python запустится в командном режиме, выглядеть это будет примерно так (картинка приведена для Windows, в Linux результат будет аналогичным):



В окне введите:

```
print("Hello, World!")
```

Результат должен быть следующий:

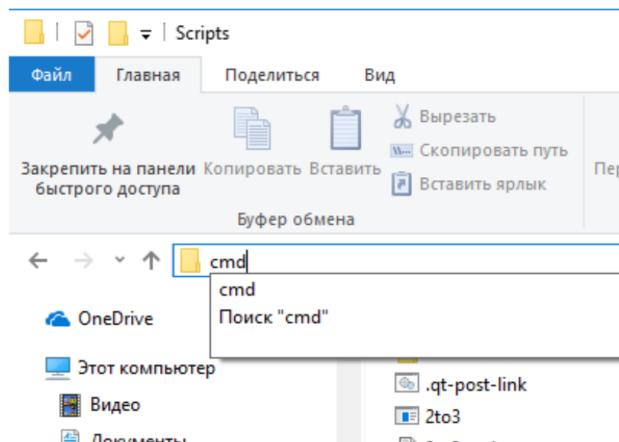


A screenshot of a Windows command prompt window titled 'F:\Program Files (x86)\Python35-32\python.exe'. The window shows the Python 3.5.2 interpreter running. The user has typed 'print("Hello, World!")' and the output 'Hello, World!' is displayed below the prompt.

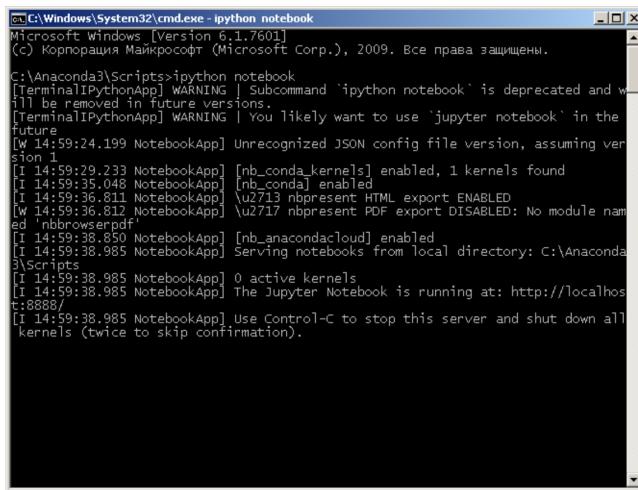
**Проверка Anaconda.** Здесь и далее будем считать, что пакет Anaconda установлен в Windows, в папку C:\Anaconda3, в Linux, вы его можете найти в каталоге, который выбрали при установке.

Перейдите в папку **Scripts** и введите в командной строке:

```
Terminal
> ipython notebook
```



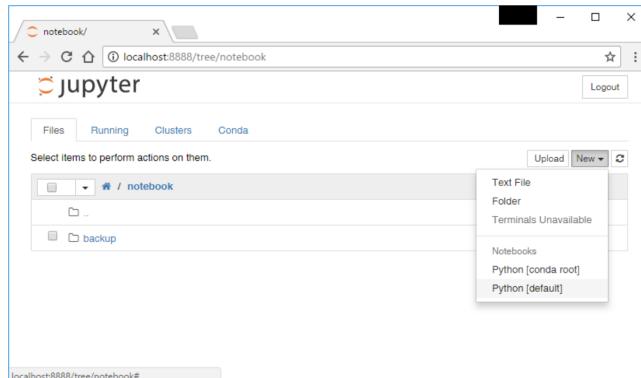
Если вы находитесь в Windows и открыли папку C:\Anaconda3\Scripts через проводник, то для запуска интерпретатора командной строки для этой папки в поле адреса введите cmd.



```
C:\Windows\System32\cmd.exe - ipython notebook
Microsoft Windows [Version 6.1.7601]
(c) Корпорация Майкрософт (Microsoft Corp.), 2009. Все права защищены.

C:\Anaconda3\Scripts>ipython notebook
[Terminal]PythonApp] WARNING | Subcommand 'ipython notebook' is deprecated and will be removed in future versions.
[Terminal]PythonApp] WARNING | You likely want to use 'jupyter notebook' in the future
[W 14:59:24.199 NotebookApp] Unrecognized JSON config file version, assuming version 1
[I 14:59:29.233 NotebookApp] [nb_conda_kernels] enabled, 1 kernels found
[I 14:59:35.048 NotebookApp] [nb_conda] enabled
[I 14:59:36.811 NotebookApp] \u2713 nbpresent HTML export ENABLED
[W 14:59:36.812 NotebookApp] \u2713 nbpresent PDF export DISABLED: No module named 'nbbrowserpdf'
[I 14:59:38.850 NotebookApp] [nb_anacondacloud] enabled
[I 14:59:38.985 NotebookApp] Serving notebooks from local directory: C:\Anaconda3\Scripts
[I 14:59:38.985 NotebookApp] 0 active kernels
[I 14:59:38.985 NotebookApp] The Jupyter Notebook is running at: http://localhost:8888
[I 14:59:38.985 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
```

В результате запустится веб-сервер и среда разработки в браузере.



Создайте ноутбук для разработки, для этого нажмите на кнопку «New» (в правом углу окна) и в появившемся списке выберете Python.

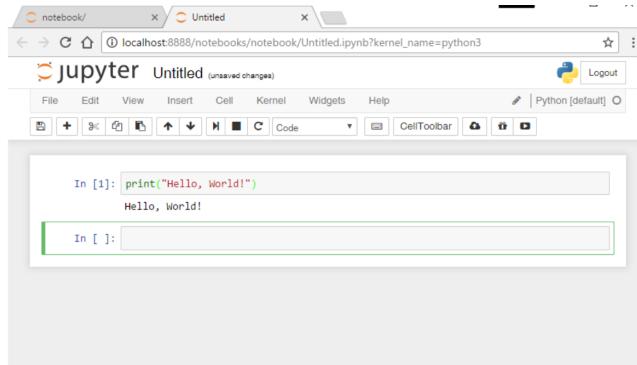
В результате будет создана новая страница в браузере с ноутбуком. Введите в первой ячейке команду

---

```
print("Hello, World!")
```

---

и нажмите Alt+Enter на клавиатуре. Ниже ячейки должна появиться соответствующая надпись.



## 2. Установка библиотек

В зависимости от вашей системы и предыдущих инсталляций среда Python может оказаться неукомплектованной всем тем, что может понадобиться.

Чтобы установить любую нужную библиотеку, можно применить команду `pip`. Инструмент установки библиотек Python `pip` непосредственно получает доступ к Интернету и получает их из каталога библиотек Python PyPI. PyPI представляет собой репозиторий, содержащий сторонние библиотеки с открытым исходным кодом, которые постоянно поддерживаются в работоспособном состоянии и сохраняются в репозитории их автором.

Устанавливать библиотеки лучше всего при помощи `pip` по следующим причинам:

- он является предпочтительным диспетчером библиотек Python и начиная с Python 2.7.9 и Python 3.4 по умолчанию включен в дистрибутивы Python;
- он обеспечивает функциональность по деинсталляции библиотек;
- он возвращает вашу систему в исходное состояние и оставляет ее чистой, если по какой-либо причине установленная библиотека перестала работать.

Команда `pip` работает в командной строке. Чтобы удостовериться в том, что инструмент `pip` установлен на локальной машине, выполните следующую команду:

```
----- Terminal -----  
> pip -V  
-----
```

В некоторых инсталляциях в Linux и Mac OS устанавливается и Python 3 и Python 2, в результате чего могут присутствовать команды `pip3` и `pip2`. Если

это так, то `pip2` подходит только для установки библиотек в Python 2, а команды `pip` и `pip3` — только для библиотек Python 3.

Если проверка закончилась ошибкой, то вам действительно нужно установить `pip` с нуля. Для установки `pip` следуйте инструкциям на <https://pip.pura.io/en/stable/installing/>. Самый безопасный путь состоит в том, чтобы скачать сценарий `get-pip.py` по прямой ссылке с `get-pip.py` и затем выполнить его при помощи следующей команды:

```
> python get-pip.py
```

После того, как вы удостоверитесь, что инструмент `pip` установлен, можно будет устанавливать дополнительные библиотеки Python. Чтобы установить типовую библиотеку `<lib>`, нужно просто выполнить команду:

```
> pip install <lib>
```

После этого библиотека `<lib>` и все библиотеки, от которых она зависит, будут скачаны и установлены.

Если вы установили дистрибутив Anaconda, то в нем для управления установкой библиотек используется инструмент `conda`.

## 2.1. Способы обновления библиотек

Как правило, может возникнуть ситуация, когда необходимо обновить библиотеку, потому что некая связанная с ней другая библиотека, т.н. зависимость, требует наличия более новой версии, либо имеется дополнительный функционал, который требуется задействовать. Для этого сначала нужно проверить версию установленной библиотеки, обратившись к атрибуту `__version__`, как показано в примере с библиотекой NumPy ниже:

```
>>> import numpy
>>> numpy.__version__ # 2 символа подчеркивания перед ним и после него
'1.9.0'
```

Далее если нужно ее обновить до более новой версии, скажем в точности до версии 1.9.2, то из командной строки можно выполнить следующую ниже команду:

```
> pip install -U numpy==1.9.2
```

Если вы просто заинтересованы в обновлении до последней доступной версии, то просто выполните команду

```
> pip install -U numpy
```

### 3. Среды для вычислений в Python

Программный код на языке Python можно записать с помощью любого простого текстового редактора, который способен загружать и сохранять текст либо в кодировке ASCII, либо UTF-8. По умолчанию предполагается, что файлы с программным кодом на языке Python сохраняются в кодировке UTF-8, надмножестве кодировки ASCII, с помощью которой можно представить практически любой символ любого национального алфавита. Файлы с программным кодом на языке Python обычно имеют расширение .py, хотя в некоторых UNIX-подобных системах (таких как Linux и Mac OS X) некоторые приложения на языке Python не имеют расширения, а программы на языке Python с графическим интерфейсом, в частности в Windows и Mac OS X, обычно имеют расширение .pyw. В этой книге все время будет использоваться расширение .py для обозначения консольных программ и модулей Python и расширение .pyw – для программ с графическим интерфейсом. Все примеры, представленные в книге, не требуют изменений для запуска в любой из платформ, поддерживаемых Python 3.

Язык Python – это *интерпретируемый* язык. Это означает, что помимо непосредственно самой программы, вам необходим специальный инструмент для её запуска. Напомним, что существуют компилируемые и интерпретируемые языки программирования. В первом случае, программа с языка высокого уровня переводится в машинный код для конкретной платформы. В дальнейшем, среди пользователей, она, как правило, распространяется в виде бинарного файла. Для запуска такой программы не нужны дополнительные программные средства (за исключением необходимых библиотек, но эти тонкости выходят за рамки нашего обсуждения). Самыми распространёнными языками такого типа являются C++ и C. Программы на интерпретируемых языках, выполняются интерпретатором и распространяются в виде исходного кода.

Ниже представим несколько сред вычислений в Python.

#### 3.1. Интерпретатор

Стандартный способ выполнения файлов со сценариями Python — запуск программ напрямую через интерпретатор с помощью команды `python`. Если в качестве аргумента команды `python` передается исходный файл, то выполняется код Python, записанный в файле:

```
Terminal
> python hello.py
Hello from Python!
```

Здесь файл `hello.py` содержит одну строку:

```
print("Hello from Python!")
```

Кроме выполнения файлов со сценариями Python интерпретатор можно использовать в качестве интерактивной консоли (REPL: Read-Evaluate-Print-Loop).

Выполнив команду `python` без аргументов, вы можете запустить интерпретатор Python в интерактивном режиме:

---

Terminal

```
> python
Python 3.8.1 (default, Jan 22 2020, 06:38:00)
[GCC 9.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

---

Здесь можно вводить код Python. При этом для каждой инструкции интерпретатор выполняет код и выводит результат на экран. Интерпретатор Python предоставляет такие возможности как история команд и базовое автодополнение команд.

### 3.2. Консоль IPython

IPython<sup>1</sup> — расширенная среда интерпретатора REPL для Python с дополнительными функциями для интерактивных и исследовательских вычислений. Например, IPython предоставляет улучшенный просмотр истории команд (также между сессиями), систему кэширования ввода и вывода, улучшенное автодополнение, более подробные и полезные отслеживания исключений и многое другое. Фактически, IPython теперь намного больше, чем расширенный интерфейс командной строки Python.

Выполнение команды `ipython` запускает интерпретатор IPython:

---

Terminal

```
> ipython
Python 3.8.1 (default, Jan 22 2020, 06:38:00)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.12.0 -- An enhanced Interactive Python. Type '?' for help.
```

---

In [1]:

---

**Кэширование ввода и вывода.** В консоли IPython подсказка ввода обозначается как `In[1]:`, а соответствующий вывод обозначается как `Out[1]:`, где число в квадратных скобках увеличивается для каждого нового ввода и вывода. Эти подсказки ввода и вывода называются **ячейками** в IPython. Значения как входных так и выходных предыдущих ячеек можно потом использовать через переменные `In` и `Out`, которые автоматически создаются IPython. Переменные `In` и `Out` — это список и словарь соответственно, которые могут быть проиндексированы индексом ячейки. Например, рассмотрим следующую сессию IPython:

---

```
In [1]: 3*3
Out[1]: 9
```

---

<sup>1</sup>Для дополнительной информации смотрите официальную страницу проекта IPython <http://ipython.org/> и официальную документацию.

```
In [2]: In[1]
Out[2]: '3*3'

In [3]: Out[1]
Out[3]: 9

In [4]: In
Out[4]: ['', '3*3', 'In[1]', 'Out[1]', 'In']

In [5]: Out
Out[5]: {1: 9, 2: '3*3', 3: 9, 4: ['', '3*3', 'In[1]', 'Out[1]', 'In', 'Out']}
```

---

Кэширование ввода и вывода часто полезно при интерактивных и исследовательских вычислениях, так как результат вычислений может быть доступен даже если он не был явно присвоен переменной.

Отметим, что когда ячейка выполнена, значение последнего выражения в ячейке ввода по-умолчанию отображается в соответствующей ячейке вывода, если выражение не является присваиванием или если значение не равно `None`. Вывод можно пресечь посредством ввода точки с запятой в конце инструкции:

---

```
In [6]: 1+2
Out[6]: 3

In [7]: 1+2;

In [8]: x=1

In [9]: x=2; x
Out[9]: 2
```

---

**Автодополнение и самоанализ объектов.** В Ipython нажатие клавиши TAB активирует автодополнение, которое отображает список символов (переменных, функций, классов и т.д.) с именами, которые доступны для дополнения того, что уже было набрано. Автодополнение в IPython является контекстным, и оно будет искать совпадающие переменные и функции в текущем пространстве имен или среди атрибутов и методов класса при вызове после имени экземпляра класса. Например, `os.<TAB>` даст список переменных, функций и классов из модуля `os`, а нажатие TAB после того, как набрали `os.w` даст список символов из модуля `os`, которые начинаются с `w`. Эта функция называется самоанализом объекта. Она очень полезна для интерактивного изучения свойств объекта Python.

**Документация.** Самоанализ объекта удобен для изучения API модулей и содержащихся в них классов и функций, и вместе со строками документации или «*docstrings*», которые обычно представлены в коде Python, он предоставляет встроенное динамическое справочное руководство для почти всех модулей, которые установлены и могут быть импортированы. Если закончить объект Python знаком

вопроса и выполнить команду, то будет выведена строка документации для объекта. Это аналогично вызову Python-функции `help`. Ввод объекта можно закончить двумя знаками вопроса. В этом случае IPython пытается отобразить более детальную документацию, включая (если возможно) исходный код. Например, для того чтобы получить описание функции `cos` из модуля `math` можно сделать следующее:

---

```
In [10]: import math  
In [11]: math.cos?  
Signature: math.cos(x, /)  
Docstring: Return the cosine of x (measured in radians).  
Type:     builtin_function_or_method
```

---

Строки документации могут быть заданы для модулей, функций, классов и их атрибутов и методов. Хорошо документированные модули включают документацию своего API в исходный код.

**Взаимодействие с оболочкой системы.** IPython предоставляет также расширение языка Python, которое делает его удобным для взаимодействия с оболочкой системы, в которой он запущен. Все, что следует за восклицательным знаком выполняется с помощью системной оболочки. Например, в UNIX-подобных системах, таких как Linux или Mac OS X, список файлов из текущего каталога можно получить следующим образом:

---

```
In [12]: !ls  
hello.py test.py
```

---

В Windows эквивалентная команда — `!dir`. Этот метод взаимодействия с операционной системой является очень мощной функцией, которая дает возможность легкой навигации по файловой системе и использования IPython в качестве консольной системной оболочки. Вывод команд, следующих за восклицательным знаком, может быть легко записан в переменную Python. Например, список файлов, полученный командой `!ls` можно сохранить в список следующим образом:

---

```
In [13]: files = !ls  
In [14]: len(files)  
Out[14]: 2  
In [15]: files  
Out[15]: ['hello.py', 'test.py']
```

---

Аналогично, мы можем передать значения переменных Python командам оболочки, поставив перед именем переменной знак `$`:

---

```
In [16]: file = "test.py"
In [17]: !ls -l $file
-rw-r--r-- 1 svl users 29 фев 17 13:59 test.py
```

---

Такое двухстороннее взаимодействие между консолью IPython и системной оболочкой может оказаться очень удобным когда, например, мы обрабатываем файлы с данными.

**Расширения IPython.** IPython предоставляет команды расширения, так называемые *магические* команды. Эти команды начинаются со знака %. Один знак % используется для односторочных команд, а два знака % используются для команд, которые действуют на нескольких ячейках (многострочные команды). Чтобы получить полный список доступных команд расширения, наберите `%lsmagic`, а документацию для каждой команды можно получить заканчивая магическую команду знаком вопроса:

---

```
In [10]: %lsmagic?
Docstring: List currently available magic functions.
File:      /usr/lib/python3.8/site-packages/IPython/core/magics/basic.py
```

---

**Навигация по файловой системе.** Кроме взаимодействия с системной оболочкой IPython предоставляет команды для навигации по файловой системе. Команды будут знакомы пользователям UNIX-подобных систем: `%ls` (список файлов), `%pwd` (вернуть текущий рабочий каталог), `%cd` (изменить рабочий каталог), `%cp` (копировать файл), `%less` (вывести содержимое файла) и `%writefile filename` (записать содержимое ячейки в файл `filename`). Отметим, что в IPython автодополнение работает и для имен файлов в текущем работочем каталоге.

**Запуск сценариев из консоли IPython.** Команда `%run` — важное и полезное расширение. С помощью этой команды можно выполнять внешние файлы со сценариями Python в интерактивной сессии IPython. Оставляя сессию активной между несколькими запусками сценария, мы можем анализировать переменные и функции, определенные в сценарии, интерактивно после окончания выполнения сценария. Рассмотрим файл

---

```
# -*- coding: utf-8 -*-
def fib(n):
    """
    Возвращает список первых n чисел Фибоначи
    """
    f0, f1 = 0, 1
    f = [1]*n
    for i in range(1, n):
        f[i] = f0 + f1
        f0, f1 = f1, f[i]
    return f
```

```
f0, f1 = f1, f[i]
return f
print(fib(10))
```

---

Сценарий определяет функцию, которая генерирует последовательность из  $n$  чисел Фибоначчи, и выводит результат при  $n = 10$ . Его можно запустить в системной консоли, используя стандартный интерпретатор Python:

---

Terminal

```
> python fib.py
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

---

Также этот файл со сценарием можно выполнить в интерактивной сессии IPython, которая выполнит тот же вывод, а также добавит символы определенные в файле в локальное пространство имен, так что функция `fib` будет доступна для использования в интерактивной сессии после выполнения команды `%run`.

---

```
In [1]: %run fib.py
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]

In [2]: %who
fib

In [3]: fib(6)
Out[3]: [1, 1, 2, 3, 5, 8]
```

---

В предыдущем примере мы также использовали команду `%who`, которая перечисляет все определенные символы (переменные и функции). Команда `%whos` аналогична, но также предоставляет более подробную информацию о типе и значении каждого символа, когда это применимо.

**Отладчик.** IPython включает режим отладки, который может быть вызван после того как возникло исключение Python (ошибка). После того, как в консоли IPython было напечатано сообщение об ошибке (Traceback) не перехваченного исключения, можно непосредственно перейти в отладчик с помощью команды `%debug`. Такая возможность позволяет не перезапускать программу с самого начала с помощью отладчика или с использованием метода отладки с добавлением операторов печати в код. Если исключение было неожиданным и произошло после длительных вычислений, это может значительно сэкономить время.

Для того, чтобы увидеть как может работать команда `%debug`, рассмотрим следующий некорректный вызов функции `fib`, определенной выше. Вызов будет некорректным, так как мы передадим значение типа `float` в то время, как функция реализована в предположении, что передаваемый аргумент имеет тип `int`.<sup>4</sup>

```
In [4]: fib(1.0)
-----
TypeError                                 Traceback (most recent call last)
<ipython-input-4-da4664af34ee> in <module>
----> 1 fib(1.0)

~/Проекты/Doconce/python-course/src/chapters/intro/src-intro/fib.py in fib(n)
    5     """
    6     f0, f1 = 0, 1
----> 7     f = [1]*n
    8     for i in range(1, n):
    9         f[i] = f0 + f1

TypeError: can't multiply sequence by non-int of type 'float'
```

В строке с номером 7 код сталкивается с ошибкой типа, и интерпретатор Python вызывает исключение типа `TypeError`. IPython перехватывает исключение и выводит полезную трассировку вызовов в консоль. Если мы не понимаем, почему код в строке 7 содержит ошибку, полезно запустить отладчик с помощью команды `%debug`. Тогда мы можем получить доступ к локальному пространству имен в источнике исключения, который позволит нам изучить более детально, почему возникло исключение.

```
In [5]: %debug
> /home/svl/Проекты/Doconce/python-course/src/chapters/intro/src-intro/fib.py(7)fib()
    5     """
    6     f0, f1 = 0, 1
----> 7     f = [1]*n
    8     for i in range(1, n):
    9         f[i] = f0 + f1

ipdb> print(n)
1.0
```

### Подсказка.

Наберите знак вопроса в строке отладчика для того, чтобы получить подсказку с перечислением доступных команд.

```
ipydb> ?
```

Больше информации об отладчике Python и его возможностях можно получить в документации Python по адресу <http://docs.python.org/3/library/pdb.html>.

**Сброс пространства имен.** Сброс пространства имен сессии IPython часто полезен для гарантии того, что программа запускается в нетронутой среде, не перегруженной существующими переменными и функциями. Команда `%reset` предоставляет такую возможность (используйте аргумент `-f` для принудительного сброса). Использование этой команды часто устраняет необходимость в других распространенных циклах выхода-перезапуска консоли. Несмотря на то, что после выполнения команды `%reset` необходимо заново импортировать модули, важно знать, что даже если модули изменились с момента последнего импорта, новый импорт после команды `%reset` не будет импортировать новый модуль, а, скорее всего, включит кэшированную версию модуля из предыдущего импорта. Такое поведение не желательно при разработке новых модулей Python. В этом случае повторный импорт ранее импортированного (и с тех пор обновленного) модуля часто может быть достигнут с помощью функции `reload` из `IPython.lib.deepreload`. Однако, этот метод не всегда работает, так как некоторые библиотеки выполняют код во время импорта, который должен выполняться только один раз. В этом случае есть только один способ: остановить и заново запустить консоль IPython.

**Расчет времени выполнения и профилирование кода.** Команды `%timeit` и `%time` предоставляют простые средства сравнения, которые полезны при попытках оптимизировать код. Команда `%timeit` запускает оператор Python несколько раз и дает оценку времени выполнения (используйте `%%timeit`, чтобы сделать то же самое для многострочной ячейки). Точное количество раз выполнения оператора, определяется эвристически, если явно не установлено с помощью флагов `-n` и `-r`. Команда `%timeit` не возвращает значение выражения. Если необходим результат выполнения вычислений, можно использовать команды `%time` или `%%time` (для многострочной ячейки), но эти команды запускают выражение только один раз и, следовательно, дают менее точную оценку времени выполнения.

Следующий пример демонстрирует типичное использование команд `%timeit` и `%time`

---

```
In [6]: %timeit fib(100)
8.15 µs ± 196 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

---

---

```
In [7]: result = %time fib(100)
CPU times: user 19 µs, sys: 0 ns, total: 19 µs
Wall time: 23.1 µs
```

---

Хотя команды `%timeit` и `%time` полезны для измерения времени выполнения вычислений, они не дают никакой подробной информации о том, какая часть вычислений занимает больше времени. Такой анализ требует более сложного профилировщика кода, такого, например, как `cProfile`, предоставляемого стандартной библиотекой Python. Профилировщик Python доступен в IPython с помощью

команд `%prun` (для операторов) и `%run` с аргументом `-p` (для запуска внешних сценариев). Вывод профилировщика довольно многословен и может быть настроен с помощью опций команд `%prun` и `%run -p` (см. `%prun?`).

В качестве примера рассмотрим функцию, которая моделирует  $N$  случайных ходоков, которые делают по  $M$  шагов, а затем вычисляет самое дальнее расстояние от начальной точки, достигнутой любым из случайных ходоков.

---

```
# -*- coding: utf-8 -*-
import numpy as np

def random_walker_max_distance(M, N):
    trajectories = [np.random.randn(M).cumsum() for _ in range(N)]
    return np.max(np.abs(trajectories))
```

---

Вызов этой функции с использованием профилировщика дает генерирует следующий вывод, который включает информацию о том, как много раз вызывается каждая функция и разбивку общего и совокупного времени, потраченного на каждую функцию. По этой информации мы можем сделать вывод, что в нашем простом примере вызовы функции `np.random.rand` потребляют большую часть времени вычислений.

---

```
In [15]: %prun random_walker_max_distance(400, 10000)
20013 function calls in 0.257 seconds

Ordered by: internal time

      ncalls  tottime  percall  cumtime  percall filename:lineno(function)
          10000   0.145    0.000    0.145    0.000 {method 'randn' of 'numpy.random.mtrand.RandomState' objects}
             1   0.056    0.056    0.254    0.254 walkers.py:4(random_walker_max_distance)
          10000   0.042    0.000    0.042    0.000 {method 'cumsum' of 'numpy.ndarray' objects}
             1   0.008    0.008    0.195    0.195 walkers.py:5(<listcomp>)
             1   0.003    0.003    0.003    0.003 {method 'reduce' of 'numpy.ufunc' objects}
             1   0.003    0.003    0.257    0.257 <string>:1(<module>)
             1   0.000    0.000    0.257    0.257 {built-in method builtins.exec}
             1   0.000    0.000    0.003    0.003 fromnumeric.py:73(_wrapreduction)
             1   0.000    0.000    0.003    0.003 <__array_function__ internals>:2(amax)
             1   0.000    0.000    0.003    0.003 fromnumeric.py:2551(amax)
             1   0.000    0.000    0.003    0.003 {built-in method numpy.core._multiarray_umath.implement_array_
             1   0.000    0.000    0.000    0.000 fromnumeric.py:74(<dictcomp>)
             1   0.000    0.000    0.000    0.000 fromnumeric.py:2546(_amax_dispatcher)
             1   0.000    0.000    0.000    0.000 {method 'items' of 'dict' objects}
             1   0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
```

---

### 3.3. Интерпретатор и текстовый редактор как среда разработки

В принципе, интерпретаторы Python и IPython и хороший текстовый редактор — это все, что нужно для полной среды разработки на Python. Эта простая связка,

по сути, является предпочтительной средой разработки для многих опытных программистов. Однако в следующих разделах мы рассмотрим Jupyter Notebook и интегрированную среду разработки Spyder. Эти среды предоставляют более широкие возможности, которые повышают производительность при работе с интерактивными и исследовательскими приложениями.

### 3.4. Jupyter

Проект Jupyter<sup>2</sup> Проект Jupyter является побочным продуктом проекта IPython, который включает независимые интерфейсы Python — в первую очередь блокноты (notebook) — и коммуникационную структуру, которая позволяет отделить внешний интерфейс от вычислительных серверных частей, известных как ядра.

До создания проекта Jupyter, блокноты и базовая структура приложения для них были частью проекта IPython. Однако, поскольку внешний интерфейс для блокнотов не зависит от языка — его также можно использовать с большим количеством других языков, таких как R и Julia, — он был выделен из отдельного проекта, чтобы лучше обслуживать более широкое вычислительное сообщество и избегать предполагаемой предвзятости в сторону Python. Теперь роль IPython заключается в том, чтобы сосредоточиться на приложениях, специфичных для Python, таких как интерактивная консоль Python, и предоставить ядро Python для среды Jupyter.

В среде Jupyter интерфейс взаимодействует с серверными частями (ядрами). Интерфейс может иметь несколько зарегистрированных ядер, например, для различных языков программирования, для различных версий Python, или различных сред Python. Ядро поддерживает состояние интерпретатора и выполняет фактические вычисления, в то время как интерфейс управляет тем, как код вводится и организуется, и как результаты вычислений визуализируются пользователю.

Здесь мы рассмотрим Jupyter QtConsole и интерфейсы для блокнотов.

**Jupyter QtConsole.** Jupyter QtConsole — это расширенное консольное приложение, которое может служить заменой стандартной консоли IPython. QtConsole запускается передачей аргумента `qtconsole` команде `jupyter`:

```
----- [Terminal] -----  
> jupyter qtconsole  
-----
```

Откроется новое приложение в системной консоли, способное отображать мультимедийные объекты, такие как рисунки и математические уравнения. Jupyter QtConsole также предоставляет механизм на основе меню для отображения результатов автозаполнения, и он запрашивает строки документации для функций во всплывающем окне при вводе открывающей скобки функции или вызова метода. Скриншот Jupyter QtConsole показан на следующем рисунке:

<sup>2</sup>Больше информации по Jupyter доступно по ссылке: <http://jupyter.org>.

The screenshot shows the Jupyter QtConsole interface. The menu bar includes File, Edit, View, Kernel, Window, and Help. The title bar says "Jupyter QtConsole". The main area displays the following session:

```

Jupyter QtConsole 4.6.0
Python 3.8.1 (default, Jan 22 2020, 06:38:00)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.12.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import sympy
In [2]: sympy.init_printing()
In [3]: x = sympy.symbols("x")
In [4]: i = sympy.Integral(x**2,(x,0,1)); i
Out[4]:

$$\int_0^1 x^2 \, dx$$


In [5]: |

```

**Jupyter Notebook.** Кроме интерактивной консоли Jupyter также предоставляет веб-приложение для блокнотов. Блокнот предоставляет много преимуществ по сравнению с традиционной средой разработки при работе с анализом данных и решением вычислительных задач. В частности, среда блокнота позволяет писать и запускать код, отображать выходные данные, произведенные кодом, а также документировать и интерпретировать код и результаты: все в одном документе. Это означает, что весь рабочий процесс анализа записывается в один файл, который впоследствии можно сохранить, восстановить и использовать повторно. Напротив, при работе с текстовым редактором или IDE код, соответствующие файлы данных и рисунки, а также документация распределяются по нескольким файлам в файловой системе, и для поддержания такого рабочего процесса требуются значительные усилия.

Jupyter Notebook оснащен богатой системой отображения, которая может отображать мультимедиа, такие как уравнения, рисунки и видео, как встроенные объекты в блокноте. Также возможно создавать элементы пользовательского интерфейса с HTML и JavaScript, используя систему виджетов Jupyter. Эти виджеты можно использовать в интерактивных приложениях, которые связывают веб-приложение с кодом Python, который выполняется в ядре IPython (на стороне сервера). Эти и многие другие функции Jupyter Notebook делают его отличной средой для интерактивных и грамотных вычислений. Для запуска среды Jupyter Notebook аргумент `notebook` передается команде `jupyter`.

---

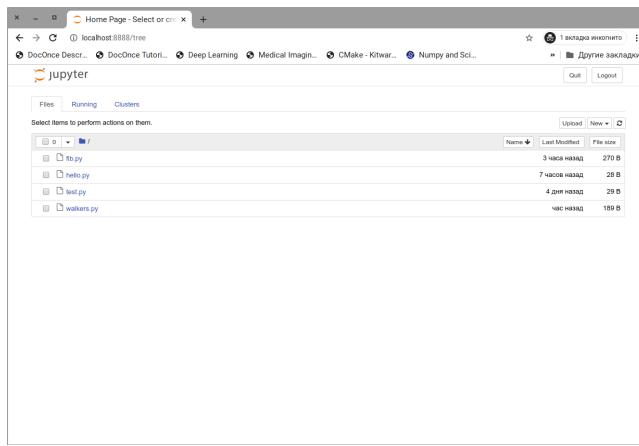
\_\_\_\_\_ Terminal \_\_\_\_\_

```
> jupyter notebook
```

---

Это запускает ядро ноутбука и веб-приложение, которое по умолчанию обслуживает веб-сервер через порт 8888 на локальном хосте, доступ к которому осу-

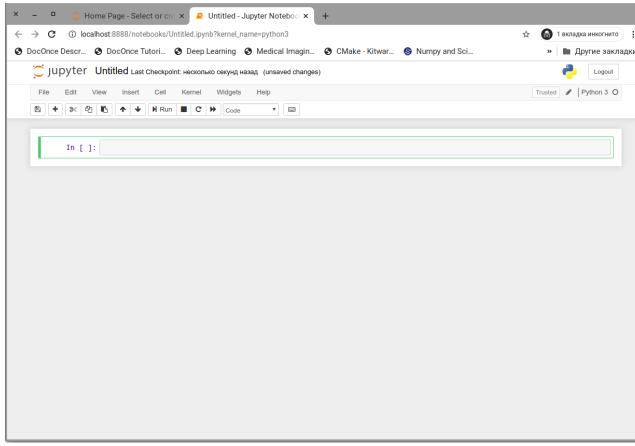
ществляется через локальный адрес <http://localhost:8888/> в веб-браузере<sup>3</sup>. По умолчанию при запуске `jupyter notebook` в браузере открывается страница с панелью инструментов. На панели инструментов перечислены все блокноты, доступные в каталоге, из которого был запущен Jupyter Notebook, а также простой обозреватель каталогов, который можно использовать для навигации по подкаталогам относительно места, где был запущен сервер, и для открытия блокнотов в нем. На рисунке показан снимок экрана веб-браузера и страницы с панелью инструментов Jupyter Notebook:



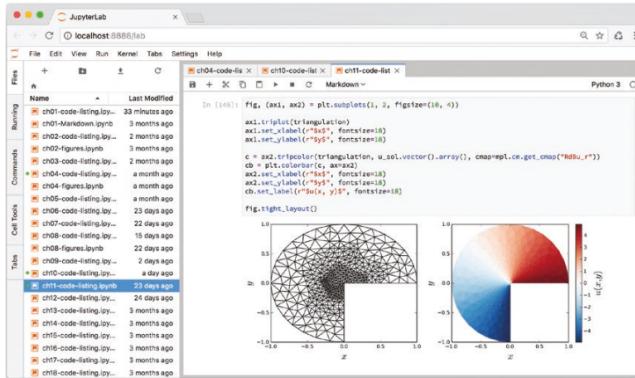
Нажатием кнопки «New» создадим новый блокнот и откроем его на новой странице в браузере. Вновь созданный блокнот имеет имя `Untitled` или `Untitled1` и т.д. Блокнот можно переименовать нажав на поле с заголовком в верхней части страницы. Файлы Jupyter Notebook сохраняются в формате JSON со специальным разрешением `iupynb`. Файл Jupyter Notebook не является чистым кодом на Python, но при необходимости код на Python можно легко выделить из блокнота используя либо меню «File □ Download as □ Python» или с помощью утилиты `nbconvert`.

---

<sup>3</sup>Это веб-приложение по умолчанию доступно только локально из системы, в которой было запущено приложение для ноутбука.



**Jupyter Lab.** Jupyter Lab — это новая альтернативная среда разработки из проекта Jupyter. Она объединяет интерфейс Jupyter Notebook с файловым браузером, текстовым редактором, командной оболочкой и консолью IPython:



Среда Jupyter Lab объединяет в себе множество преимуществ среды для блокнотов и сильные стороны традиционных IDE. Доступ к консолям оболочки и текстовым редакторам в одном веб-интерфейсе также удобен при работе на сервере Jupyter, работающем в удаленной системе, например в вычислительном кластере или в облаке.

**Типы ячеек.** Основное содержимое блокнота, расположенное под строкой меню и панелью инструментов, организовано как ячейки ввода и вывода. Ячейки могут быть нескольких типов, тип выбранной ячейки можно изменять используя меню в панели инструментов. Наиболее важные типы:

- **Code:** ячейка с кодом может содержать произвольное количество многострочного кода на Python. Нажатие «Shift-Enter» посыпает код из ячейки ядру для выполнения в интерпретаторе Python. Результат выполнения возвращается браузеру и отображается в соответствующей ячейке вывода.
- **Markdown:** ячейка такого типа может содержать размеченный текст, который интерпретируется с использованием языка Markdown и HTML.
- **Headings:** ячейка такого типа может использоваться для разбивки блокнота на разделы.
- **Raw:** ячейка такого типа отображает необработанный текст.

**Редактирование ячеек.** Используя меню и панель инструментов, ячейки можно добавлять, удалять, перемещать вверх и вниз, вырезать и вставлять и т.д. Эти функции также связаны с сочетаниями клавиш, которые удобны и экономят время при работе с ноутбуками Jupyter. В ноутбуке используется двухрежимный интерфейс ввода: режим редактирования и командный режим. В режим редактирования можно войти, щелкнув ячейку или нажав клавишу ввода на клавиатуре, когда ячейка находится в фокусе. Находясь в режиме редактирования, содержимое ячейки ввода можно редактировать. Выход из режима редактирования осуществляется нажатием клавиши ESC или нажатием Shift-Enter для выполнения ячейки. В командном режиме стрелки вверх и вниз можно использовать для перемещения фокуса между ячейками, а ряд сочетаний клавиш сопоставляется с основными действиями по манипулированию ячейками, доступными через панель инструментов и меню.

Во время выполнения ячейки блокнота номер ячейки обозначается звездочкой (`In [ * ]`), а индикатор в правом верхнем углу страницы сигнализирует о том, что ядро IPython занято. Прервать выполнение ячейки можно через меню «Kernel □ Interrupt» или сочетанием клавиш `i - i` в командном режиме (т.е., дважды нажать клавишу `i`). В таблице 1 представлены наиболее популярные сочетания клавиш командного режима Jupyter Notebook.

**Ячейки Markdown.** Одна из ключевых возможностей Jupyter Notebook заключается в том, что ячейки с кодом и ячейки с результатом могут быть дополнены документацией, содержащейся в текстовых ячейках. Текстовые ячейки имеют тип Markdown. Входной текст преобразуется с использованием языка разметки Markdown. Markdown разработан как легкая система набора текста, которая позволяет преобразовывать текст с простыми правилами разметки в HTML и другие форматы для более удобного отображения. Правила разметки разработаны так, чтобы быть удобными для пользователя и читаемыми, как в текстовом формате. Например, фрагмент текста можно выделить курсивом, окружив его звездочками (`*text*`), можно сделать полужирным, окружив двойными звездочками (`**text**`). Markdown также позволяет создавать нумерованные и маркированные списки, таблицы и гиперссылки. Расширение Markdown, поддерживаемое

**Таблица 1.**

Сочетание клавиш	Описание
b	Создать ячейку ниже текущей
a	Создать ячейку выше текущей
d-d	Удалить текущую ячейку
от 1 до 6	Ячейка с заглавием раздела (heading) уровня от 1 до 6
x	Вырезать текущую ячейку в буфер обмена
c	Скопировать текущую ячейку в буфер обмена
v	Вставить ячейку из буфера обмена
m	Преобразовать тип ячейки в Markdown
y	Преобразовать тип ячейки в Code
Up	Выбрать предыдущую ячейку
Down	Выбрать следующую ячейку
Enter	Войти в режим редактирования ячейки
Escape	Выйти из режима редактирования ячейки
Shift-Enter	Выполнить ячейку
h	Показать окно помощи со списком сочетаний клавиш
0-0	Перезапустить ядро
i-i	Прервать выполнение ячейки
s	Сохранить блокнот

Jupyter, заключается в том, что математические выражения можно набирать в  $\text{\LaTeX}$  с помощью библиотеки JavaScript  $\text{\LaTeX}$  MathJax.

Ниже приведем краткое описание синтаксиса Markdown.

**Выделение.** Одинарные звездочки используются для выделения курсивом

---

это \*выделенный\* текст

---

Двойные звездочки используются для выделения полужирным

---

---

это \*\*очень выделенный\*\* текст

---

**Код в строке.**

---

Используйте функцию `printf()` для вывода.

---

**Ссылки.** Инлайн ссылки

---

Это простая [ссылка](<http://rukeba.com/>) в тексте.  
Это [ссылка](<http://rukeba.com/> "Титул") с атрибутом title.

---

## Сноски

---

Это [ссылка][1] в справочном стиле.  
...  
...  
[1]: <http://rukeba.com/> "Необязательный титул"

---

Чтобы веб-адрес в тексте стал ссылкой его нужно обрамить < и >

---

По ссылке <<http://python.org>> находится сайт.

---

## Изображения.

---

![alt-текст](<http://example.com/image.jpg> "Необязательный титул")

---

или в справочном стиле:

---

![alt-текст][2]  
...  
...  
[2]: <http://example.com/image.jpg> "Необязательный титул"

---

## Абзац.

---

<пустая строка>  
Текст параграфа. Может быть  
разбит на несколько строк.  
<пустая строка>

---

**Разрыв строки.** Что бы вставить перевод строки без начала нового параграфа, в конец нужно добавить два или больше пробелов.

---

Текст параграфа.<пробел><пробел>  
Этот текст будет с новой строки.

---

**Код.** Отступ в четыре пробела или один таб генерирует блок кода:

---

<пробел><пробел><пробел><пробел>print 'Hello, World!'

---

## **Горизонтальная линия.**

---

\* \* \*

---

или так:

---

\*\*\*

---

или так:

---

-----

---

## **Таблица.**

---

A	B	C
1	2	3
4	5	6

---

## **Заголовки.**

---

# Заголовок первого уровня H1

---

## **Цитаты.**

- 
- > Это
  - > многострочная
  - > цитата
- 

**Списки.** Ненумерованный:

- 
- \* красный
  - \* зеленый
  - \* синий
- 

Нумерованный:

- 
1. Первый
  2. Второй
  3. Третий
-

**LaTeX.** Строчные формулы:

---

```
$\LaTeX$
```

---

Выключенная формула (с новой строки и по-центру):

---

```
$$
\LaTeX
$$
```

---

или

---

```
\begin{env}
...
\end{env}
```

---

Здесь `env`  $\text{\LaTeX}$  окружение для формул: `equation`, `eqnarray`, `align` и т.д.

Ячейки типа Markdown могут также содержать код HTML, и интерфейс Jupyter Notebook будет отображать его как обработанный HTML.

Больше информации о MathJax и Markdown доступно на страницах проектов: <http://www.mathjax.com> и <http://daringfireball.net/projects/markdown/>.

### 3.5. Spyder: Интегрированная среда разработки

Рассмотрим свободную интегрированную среду разработки Spyder, особенно хорошо подходит для вычислений и анализа данных с использованием Python.

Spyder IDE был специально создан для программирования на Python и, в частности, для научных вычислений с использованием Python. Как таковой, он обладает функциями, которые полезны для интерактивных и исследовательских вычислений: в частности, интеграция с консолью IPython непосредственно в IDE. Пользовательский интерфейс Spyder состоит из нескольких дополнительных панелей, которые могут быть по-разному организованы в приложении.

Самые важные панели:

- Редактор исходного кода
- Консоли интерпретаторов Python и IPython и системной оболочки
- Инспектор объектов для показа документации по объектам Python
- Обозреватель переменных
- Обозреватель файловой
- История команд
- Профилировщик

Каждую панель можно настроить для отображения или скрытия в зависимости от предпочтений и потребностей пользователя, используя меню «View □ Panes».

**Редактор исходного кода.** Редактор исходного кода в Spyder поддерживает подсветку синтаксиса, умное автодополнение, работу с несколькими открытыми файлами одновременно, проверку соответствия скобок и др. Дополнительным преимуществом использования IDE является то, что код может быть запущен из редактора — код целиком (клавиша F5) или выделенный код (клавиша F9) — в подключенных консолях Python и IPython.

Кроме того, редактор Spyder имеет очень полезную поддержку статической проверки кода с помощью `pylint`, `pyflakes` и `pep8` — внешних инструментов, которые анализируют исходный код Python и сообщают об ошибках, таких как неопределенные символы, синтаксические ошибки, нарушения стиля кодирования и т. д.

#### Подсказка.

Язык Python является универсальным, и эквивалентный исходный код Python может быть написан с использованием самых разных стилей и способов. Однако был предложен стандарт стиля кодирования Python, PEP8, для поощрения единообразного внешнего вида кода Python. Настоятельно рекомендуем изучить стандарт стиля кодирования PEP8 и соответствовать этому в вашем коде. PEP8 описан по адресу <http://www.python.org/dev/peps/pep-0008>.

**Консоли.** Интегрированные консоли Python и IPython могут использоваться для запуска файла, который редактируется в окне текстового редактора, или для запуска кода Python с интерактивным вводом. При выполнении файлов исходного кода Python из редактора переменные пространства имен, созданные в сценарии, сохраняются в сеансе IPython или Python в консоли. Это важная особенность, которая делает Spyder интерактивной вычислительной средой, в дополнение к традиционному приложению IDE, поскольку она позволяет исследовать значения переменных после завершения выполнения сценария. Spyder поддерживает одновременное открытие нескольких консолей Python и IPython, и, например, новую консоль IPython можно запустить через меню «Consoles □ Open an IPython console». При запуске сценария из редактора нажатием клавиши F5 или кнопки запуска на панели инструментов сценарий по умолчанию запускается в самой последней активированной консоли. Это позволяет поддерживать разные консоли с независимыми пространствами имен для разных сценариев или проектов.

**Инспектор объектов.** Инспектор объектов (панель справки) очень помогает при написании кода Python. Он может отображать строки форматированного документа для объектов, определенных в исходном коде, созданном с помощью редактора, и для символов, определенных в библиотечных модулях, установленных в системе. Текстовое поле объекта в верхней части панели инспектора объектов можно использовать для ввода имени модуля, функции или класса, для которого отображается строка документации. Модули и символы не нужно импортировать

в локальное пространство имен, чтобы можно было отображать их строки документов с помощью инспектора объектов. Документацию для объекта в редакторе или консоли также можно открыть в инспекторе объектов, выбрав объект с помощью курсора и используя сочетание клавиш **Ctrl-i** (**Cmd-i** в Mac OS X). Можно даже автоматически отображать строки документов для вызываемых объектов, когда вводится его открывающая левая скобка. Это дает немедленное напоминание об аргументах и их порядке для вызываемого объекта, что может значительно повысить производительность. Чтобы активировать эту функцию, перейдите на страницу «Help» в окне «Preferences» и установите флагки в разделе «Automatic connections».