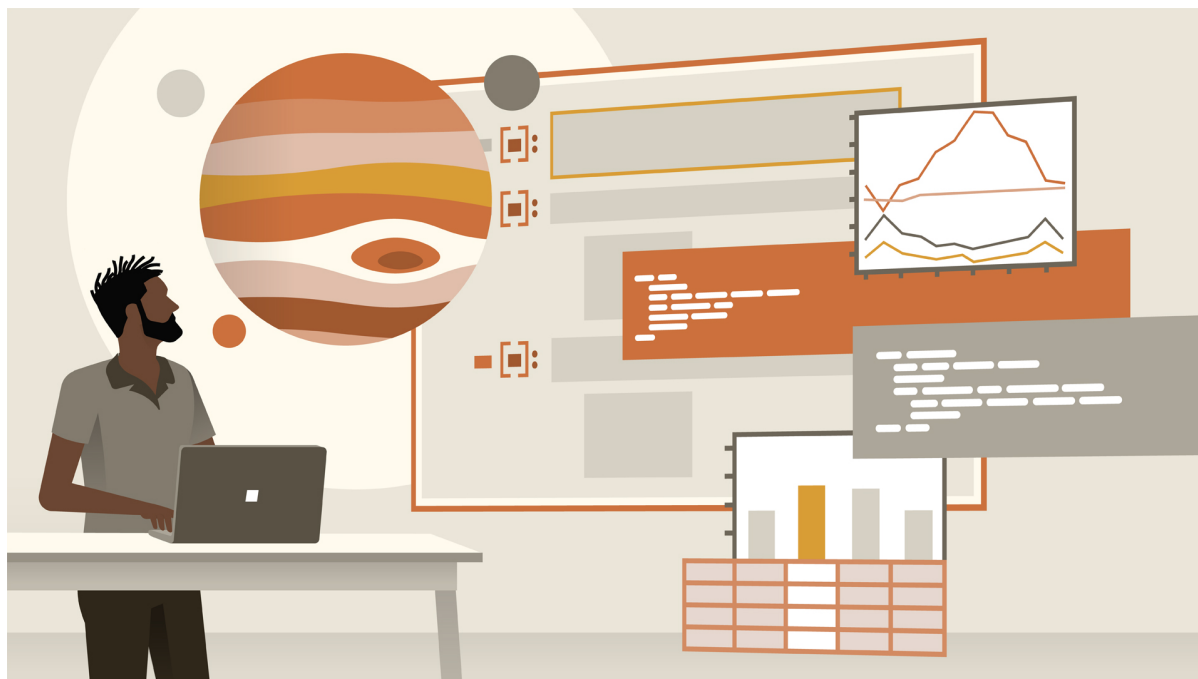


О чём этот ноутбук

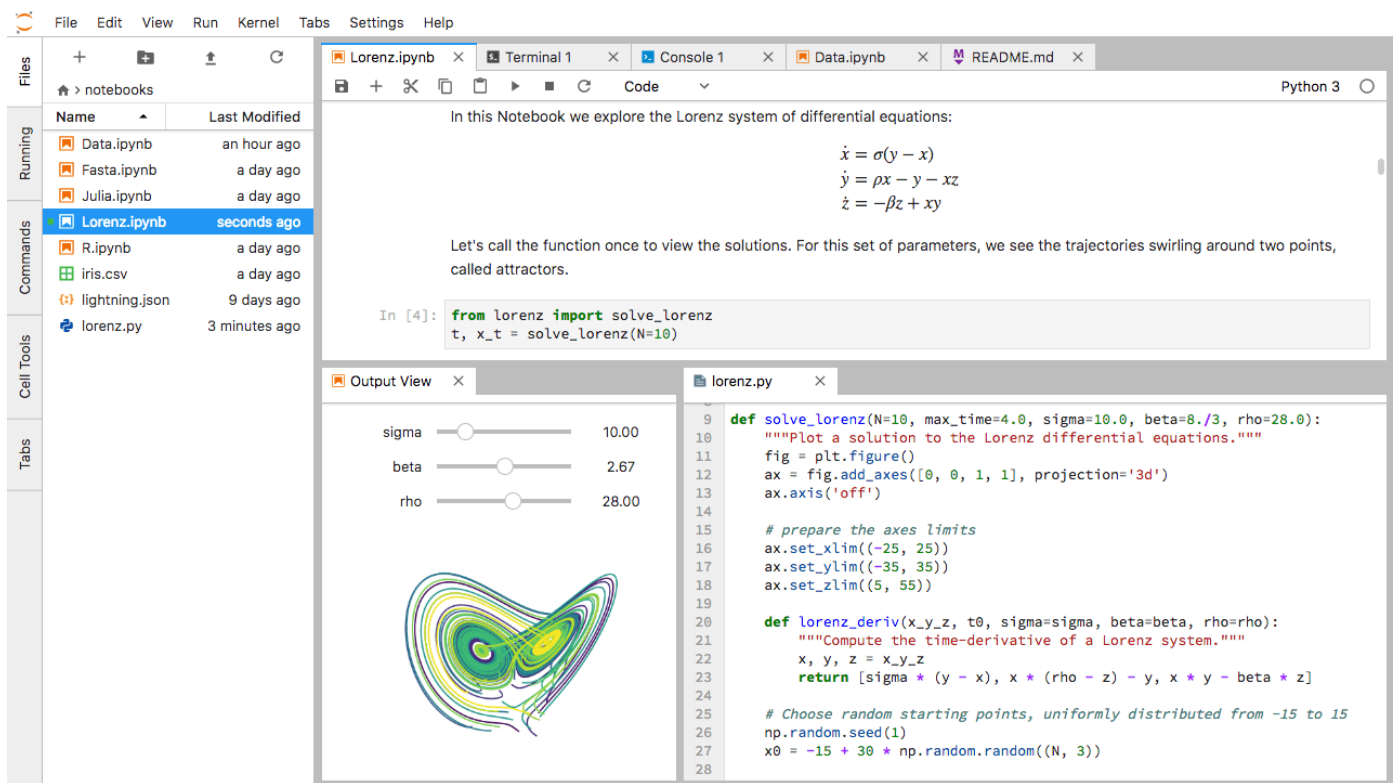
В ноутбуке есть интерактивный контент, в связи с чем мы рекомендуем просматривать его в `jupyter notebook/lab`.



["Why Jupyter is data scientists' computational notebook of choice" \(https://www.nature.com/articles/d41586-018-07196-1\)](https://www.nature.com/articles/d41586-018-07196-1) — статья с таким заголовком вышла в октябрьском номере Nature в 2018 году (почитайте, там интересно). Она посвящена Jupyter Notebook — open-source веб-приложению для интерактивной разработки, которое за последние несколько лет покорило сердца миллионов и стало де-факто стандартом в аналитике, науке о данных, машинном обучении и много где ещё. Количество .ipynb-файлов на GitHub выросло с $2 \cdot 10^5$ в 2015 году до $2.5 \cdot 10^6$ в 2018. Ниже мы разберём основные причины такой взрывной популярности:

Лабораторный журнал 21 века

Работа аналитика или учёного отличается от индустриальной разработки отсутствием чёткого техзадания. В этих сферах правильная постановка задачи это уже половина решения. На начальную фазу — тщательный анализ исходных данных (exploratory data analysis (EDA)) и последующую постановку гипотез — уходит много времени. Как следствие, до самого конца неясно, как организовать процесс разработки: что в итоге важно, а о чём можно будет забыть. В таких условиях тратить силы на скрупулёзное написание кода бессмысленно — на первый план выходит умение быстро прототипировать, ставить эксперименты. Кроме того, свои результаты приходится постоянно презентовать как коллегам, там и заказчику или начальству. Jupyter Notebook убивает этих двух зайцев одним выстрелом:



По своей сути Jupyter это интерактивный лабораторный журнал: он позволяет писать код прямо в отчёте (markdown с формулами из latex), дополнять его иллюстрациями (в том числе интерактивными), пересчитывать всё на лету и обмениваться результатами в читаемом формате. Это дало новую жизнь парадигме **reproducible research** — в формате .ipynb или R-Markdown пишут [статьи](https://github.com/jupyter/jupyter/wiki/A-gallery-of-interesting-Jupyter-Notebooks#reproducible-academic-publications) (<https://github.com/jupyter/jupyter/wiki/A-gallery-of-interesting-Jupyter-Notebooks#reproducible-academic-publications>) и даже книги (<https://github.com/jupyter/jupyter/wiki/A-gallery-of-interesting-Jupyter-Notebooks#entire-books-or-other-large-collections-of-notebooks-on-a-topic>), где заинтересованный читатель может сразу же воспроизводить описанные результаты. Автор не уверен, что впечатляет его больше — потенциальный полёт человечества к Юпитеру или то, насколько более демократичной Jupyter Notebook сделал высокую науку (которая, к слову, страдает от того, что [70% учёных не смогли воспроизвести чужие результаты, а ещё 50% — свои](https://www.nature.com/news/1-500-scientists-lift-the-lid-on-reproducibility-1.19970) (<https://www.nature.com/news/1-500-scientists-lift-the-lid-on-reproducibility-1.19970>) согласно опросу в Nature).

Нелинейная последовательность исполнения

Jupyter позволяет запускать клетки с кодом в произвольном порядке и, в отличие от IPython, в нём это делать удобно. Да, это повышает требования к разработчику (который должен помнить, какая конкретно последовательность ячеек решает задачу) и провоцирует писать очень плохой код, но в реалиях, когда полный перезапуск пайплайна требует нескольких часов на суперкомпьютере, это не прихоть, а необходимость.

Вавилонская башня от программирования

Вопреки расхожему мнению, Jupyter поддерживает не только Julia, Python и R (Ju - Py - teR, именно потому), но [более 40 языков](https://jupyter.org) (<https://jupyter.org>), включая C++ (да-да, компилируемый язык, [check it out](https://quantstack.net/index.html) (<https://quantstack.net/index.html>)). Более того, для того, чтобы писать код на нескольких языках одновременно (обычно это Bash, Python и R) не нужно создавать разные ноутбуки (об этом мы поговорим ниже). Это инструмент, который не ограничивает возможности разработчика.

Переносимость и платформонезависимость

Jupyter позволяет развернуть дружелюбный интерфейс на любой удалённой машине, от учебного сервера до гигантской распределённой `map-reduce` сети в духе Яндексовского `YT`. Учитывая, что работать с данными на своём ноутбуке почти никогда не приходится (либо по закону нельзя, либо в оперативку не влезает), возможность оставить в прошлом сумрачную серверную консоль многие восприняли с облегчением. В сочетании с инструментами в духе `conda`, `jupyter` позволяет развернуть привычное и настроенное под себя окружение, не задумываясь о том, с каким железом и в какой системе приходится работать. Более того, `Jupyter Hub` позволяет создать точку входа, через которую над проектом могут совместно работать несколько человек. Это позволяет специалисту настроить окружение один раз, после чего им смогут пользоваться сотни человек. Неудивительно, что многие онлайн-курсы переходят на формат `jupyter`-ноутбуков. Возможно, нас это тоже ждёт.

Вывод

`Jupyter`-ноутбуки это мощный инструмент, который облегчит вашу жизнь. Тем не менее, работа с ним требует дисциплины. Мы разберём как `best practices`, так и распространённые антипаттерны, которых следует избегать. Мы верим, что это сделает мир лучше.

Мы разберём такие темы как:

1. `jupyter_contrib_nbextensions` — плагины, которые превращают `jupyter` в удобный кодовый редактор;
2. основы работы с командной строкой (здесь `jupyter` похож на `vim`);
3. `jupyter magics` — метаязык, который позволяет навешивать триггеры на клетки и интерактивно менять переменные окружения. Если простыми словами, то эта штука позволяет засекать время исполнения клеток, менять формат отображения графиков, запускать внешние скрипты и многое другое, что обычно делают через терминал. Очень удобно;
4. запуск `jupyter`-сервера на удалённой машине;
5. полезные мелочи, которые сделают ваши ноутбуки опрятнее;

Основы

Установка всего необходимого в `conda-env` обсуждалась в прошлом ноутбуке.

Для того, чтобы поднять `jupyter`-сервер, активируйте `conda-env` и наберите в терминале:

```
jupyter notebook
```

Когда вы откроете `jupyter`-ноутбук, в меню `Help` можно будет найти интерактивные вводные руководства.

Советуем ознакомиться с ними, если интерфейс вам вновинку.

Всего ячейки бывают трёх типов:

- `code` — собственно, код
- `markdown` — текст с формулами на `latex` (в одинарных или двойных `$`)
- `raw` — неформатированный текст

Изменить тип можно либо в меню (`Cell` → `Cell Type`), либо с помощью клавиатурных сокращений (см. ниже).

В целом, базовый интерфейс довольно интуитивный, в нём полезно разобраться самостоятельно.

Плагины

Встроенный редактор довольно удобный, но с плагинами он становится по-настоящему хорош.

Установка библиотеки `jupyter_contrib_nbextensions`, которая отвечает за поддержку плагинов, освещалась в прошлом ноутбуке.

Давайте разберёмся, какие плагины из стандартного списка наиболее полезны:

- `Ruler` — отображает красным границу в 80 символов в строке. **Этот плагин нужно поставить обязательно**, чтобы не расстраивать проверяющих тем, что при конвертации в pdf у вас опять что-то не влезло в одну строку.
- `Codefolding`, `Codefolding in editor` — незаменимый плагин, который позволяет сворачивать код для лучшей читаемости.
- `ExecuteTime` — автоматически запоминает, когда ячейка начала выполняться и сколько времени это заняло
- `Notify` — посылает push-уведомление в браузере, когда ячейка закончила выполняться (если она обрабатывала достаточно долго)
- `Move selected cells` — позволяет одновременно перемещать выделенные ячейки вверх или вниз
- `Limit Output` — ограничивает размер вывода ячейки. С этим плагином всё не упадёт, если из-за ошибки в коде вы решите распечатать многогигабайтную таблицу прямо в окошке браузера (бывает чаще, чем хотелось бы).

Этим список плагинов не исчерпывается, остальные можете выбрать на свой вкус.

Работа с командной строкой Jupyter

Command mode

В jupyter есть `command mode` — режим, где можно вводить команды для быстрого редактирования текста. Как и в `vim`, из которого эта концепция была позаимствована, его можно активировать кнопкой `Esc`, после чего возможны варианты. `Esc + h` откроет справку:

Command Mode (press `Esc` to enable)

Edit Shortcuts

<code>F</code> : find and replace	<code>A</code> : insert cell above
<code>Ctrl-Shift-F</code> : open the command palette	<code>Alt-F</code> : Toggle codefolding
<code>Ctrl-Shift-P</code> : open the command palette	<code>B</code> : insert cell below
<code>Enter</code> : enter edit mode	<code>X</code> : cut selected cells
<code>P</code> : open the command palette	<code>C</code> : copy selected cells
<code>Shift-Enter</code> : run cell, select below	<code>Shift-V</code> : paste cells above
<code>Ctrl-Enter</code> : run selected cells	<code>V</code> : paste cells below
<code>Alt-Enter</code> : run cell and insert below	<code>Z</code> : undo cell deletion
<code>Y</code> : change cell to code	<code>D</code> , <code>D</code> : delete selected cells
<code>M</code> : change cell to markdown	<code>Shift-M</code> : merge selected cells, or current cell with cell below if only one cell is selected
<code>R</code> : change cell to raw	<code>Ctrl-S</code> : Save and Checkpoint
<code>1</code> : change cell to heading 1	<code>S</code> : Save and Checkpoint
<code>2</code> : change cell to heading 2	<code>L</code> : toggle line numbers
<code>3</code> : change cell to heading 3	<code>O</code> : toggle output of selected cells
<code>4</code> : change cell to heading 4	<code>Shift-O</code> : toggle output scrolling of selected cells
<code>5</code> : change cell to heading 5	<code>H</code> : show keyboard shortcuts
<code>6</code> : change cell to heading 6	<code>I</code> , <code>I</code> : interrupt the kernel
<code>K</code> : select cell above	<code>0</code> , <code>0</code> : restart the kernel (with dialog)
<code>Up</code> : select cell above	<code>Ctrl-L</code> : code_prettify selected cell(s)
<code>Down</code> : select cell below	<code>Ctrl-Shift-L</code> : code_prettify the whole notebook
<code>J</code> : select cell below	<code>Esc</code> : close the pager
<code>Shift-K</code> : extend selected cells above	<code>Q</code> : close the pager
<code>Shift-Up</code> : extend selected cells above	<code>Shift-L</code> : toggles line numbers in all cells, and persist the setting
<code>Shift-Down</code> : extend selected cells below	<code>Shift-Space</code> : scroll notebook up
<code>Shift-J</code> : extend selected cells below	<code>Space</code> : scroll notebook down

В ней перечислено много команд, из которых чаще всего используются такие:

- `Esc + F` — найти и заменить
- `Esc + I` — прервать исполнение
- `Esc + 0` — перезапустить ядро
- `Esc + Shift + ↑↓` — выделить ячейку выше/ниже текущей
- `Esc + Ctrl + Enter` — запустить выделенные ячейки (в естественном порядке)
- `Esc + D + D` — удалить все выделенные ячейки (включая текущую)
- `Esc + Z` — отменить удаление ячеек
- `Esc + C` — скопировать выделенные ячейки
- `Esc + X` — вырезать выделенные ячейки

- Esc + V — вставить ранее скопированные ячейки
- Esc + A — вставить ячейку выше
- Esc + B — вставить ячейку ниже
- Esc + Shift + M — объединить выделенные ячейки (или текущую ячейку с нижней)
- Esc + M — заменить код на markdown
- Esc + Y — наоборот, markdown —> код

Edit mode

Эти сокращения доступны в редакторе. Здесь всё стандартно. Если вы не привыкли пользоваться такими сокращениями, то настоятельно рекомендуем потратить время и научиться: это сэкономит вам много времени в будущем.

Edit Mode (press `Enter` to enable)

<code>Tab</code> : code completion or indent	<code>Ctrl-Delete</code> : delete word after
<code>Shift-Tab</code> : tooltip	<code>Ctrl-Y</code> : redo
<code>Ctrl-]</code> : indent	<code>Alt-U</code> : redo selection
<code>Ctrl-[</code> : dedent	<code>Ctrl-M</code> : enter command mode
<code>Ctrl-A</code> : select all	<code>Ctrl-Shift-F</code> : open the command palette
<code>Ctrl-Z</code> : undo	<code>Ctrl-Shift-P</code> : open the command palette
<code>Ctrl-/</code> : comment	<code>Esc</code> : enter command mode
<code>Ctrl-D</code> : delete whole line	<code>Shift-Enter</code> : run cell, select below
<code>Ctrl-U</code> : undo selection	<code>Ctrl-Enter</code> : run selected cells
<code>Insert</code> : toggle overwrite flag	<code>Alt-Enter</code> : run cell and insert below
<code>Ctrl-Home</code> : go to cell start	<code>Ctrl-Shift-Minus</code> : split cell at cursor
<code>Ctrl-Up</code> : go to cell start	<code>Alt-F</code> : Toggle codefolding
<code>Ctrl-End</code> : go to cell end	<code>Ctrl-S</code> : Save and Checkpoint
<code>Ctrl-Down</code> : go to cell end	<code>Ctrl-L</code> : code_prettify selected cell(s)
<code>Ctrl-Left</code> : go one word left	<code>Ctrl-Shift-L</code> : code_prettify the whole notebook
<code>Ctrl-Right</code> : go one word right	<code>Down</code> : move cursor down
<code>Ctrl-Backspace</code> : delete word before	<code>Up</code> : move cursor up

Самые полезные команды (кроме очевидных) таковы:

- Shift + Tab — показать описание функции (курсор должен стоять сразу после открывающей скобки)
- Ctrl + / — закомментировать строку
- Ctrl + D — удалить строку
- Ctrl + ↑ — перейти к началу ячейки
- Ctrl + ↓ — перейти к концу ячейки
- Ctrl + Shift + - — разбить ячейку по текущей строке
- Shift + ↑↓ — выделить ячейку выше/ниже текущей
- Ctrl + Enter — запустить выделенные ячейки
- Shift + Enter — запустить ячейку и перейти на одну ниже (удобно, когда последовательно запускаете сразу много)

Магия Jupyter



`jupyter magics` — метаязык (как макросы в Си), команды которого обычно начинаются с `%` или `%%` (зависит от контекста). Работают они или нет — зависит от реализации. Стандартные `jupyter-magics` поставляются с ядром IPython (потому вы можете выучить их один раз, а потом пользоваться и там, и там). Для других языков или ядер набор команд может быть другим.

Давайте посмотрим, какие команды есть.

P.S. Если вам захочется узнать больше, можете посмотреть [замечательный доклад с PyCon Canada \(https://www.youtube.com/watch?v=zxkdO07L29Q\)](https://www.youtube.com/watch?v=zxkdO07L29Q).

In [58]:

```
1 %lsmagic
```

Out[58]:

Available line magics:

```
%aimport %alias %alias_magic %autoawait %autocall %automagic %autoreload %autosave %bookmark %cat %cd %clear %colors %conda %config %connect_info %cp %debug %dhist %dirs %doctest_mode %edit %env %gui %hist %history %killbgscripts %ldir %less %lf %lk %ll %load %load_ext %loadpy %logoff %logon %logstart %logstate %logstop %ls %lsmagic %lx %macro %magic %man %matplotlib %mkdir %more %mv %notebook %page %pastebin %pdb %pdef %pdoc %pfile %pinfo %pinfo2 %pip %popd %pprint %precision %prun %psearch %psource %pushd %pwd %pycat %pylab %qtconsole %quickref %recall %rehashx %reload_ext %rep %rerun %reset %reset_selective %rm %rmdir %run %save %sc %set_env %store %sx %system %tb %time %timeit %unalias %unload_ext %who %who_ls %whos %xdel %xmode
```

Available cell magics:

```
%%! %%HTML %%SVG %%bash %%capture %%debug %%file %%html %%java script %%js %%latex %%markdown %%perl %%prun %%pypy %%python %%python2 %%python3 %%ruby %%script %%sh %%svg %%sx %%system %%time %%timeit %%writefile
```

Automagic is ON, % prefix IS NOT needed for line magics.

А если вы хотите почитать краткую справку по IPython (по сути, краткий конспект этого раздела),

используйте команду

In [59]:

```
1 %quickref
```

Видно, что команды делятся на два типа: `line magics` и `cell magics`. Как следует из названия, разница между ними в том, что `cell magics` влияют на состояние всей ячейки (и работают только если написаны в самом её начале), в то время как `line magics` могут быть написаны в любом месте и либо влияют на строку с кодом (скажем, замеряют время исполнения, как `%timeit`), либо выполняют какую-то системную команду (как `%ls`). Если вы забыли, что делает конкретная команда, то добавьте перед ней знак вопроса — откроется справка.

In [2]:

```
1 ?%time
```

К слову, таким же образом можно вызвать справку для произвольной функции. Синтаксис привычен тем, кто работал с R :

In [6]:

```
1 ?range
```

Давайте посмотрим на эти команды поближе:

%load_ext, %autoreload

`%load_ext` позволяет загружать сторонние модули (например, `Cython`). Нам важно, что с её помощью можно активировать самую важную команду — `%autoreload`. Без неё вам пришлось бы перезапускать ядро каждый раз, когда вы меняете библиотеки со своим кодом. `autoreload` делает это за вас, в больших проектах это незаменимо.

In [71]:

```
1 %load_ext autoreload
2 # здесь 2 означает, что все библиотеки будут подгружаться заново,
3 # если только явно не указано иное
4 %autoreload 2
5
6 import jupyter_lesson
```

The autoreload extension is already loaded. To reload it, use:
`%reload_ext autoreload`

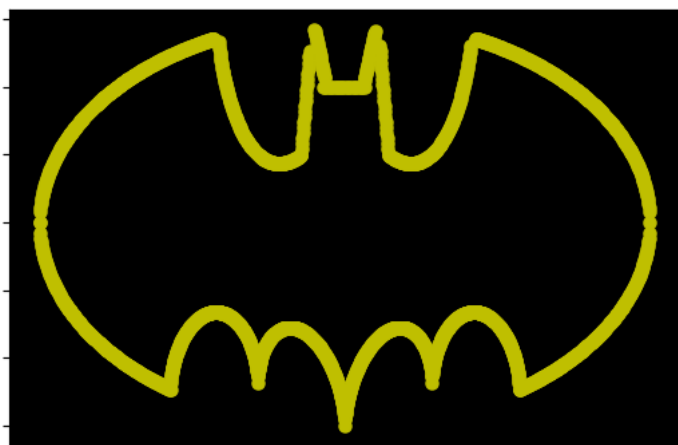
%matplotlib

Скорее всего, эту команду вам придётся писать в начале каждого домашнего задания. Она определяет, каким образом будут отображаться ваши графики. Есть два основных варианта: `inline` и `notebook`. Разберёмся, в чём между ними разница:

In [76]:

```
1 %matplotlib notebook
2
3 jupyter_lesson.plot_batman()
```

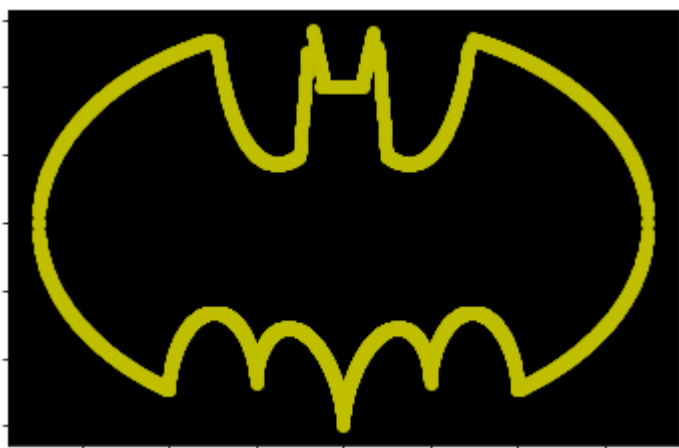
<IPython.core.display.Javascript object>



Видно, что открылось интерактивное окошко, в котором можно масштабировать участки графика по своему усмотрению. Это удобно, когда хочется рассмотреть детали, но требовательно к ресурсам при сложных визуализациях. А ещё работает не всегда.

In [75]:

```
1 %matplotlib inline
2
3 jupyter_lesson.plot_batman()
```



В этом режиме в ноутбук просто вставляется статическая картинка. Именно этим режимом вы будете пользоваться чаще всего.

Раздел со звёздочкой: На удалённых машинах часто нет графического интерфейса. В связи с этим набор предустановленных инструментов для работы с графикой там ограничен. Часто обнаруживается только какой-то конкретный бэкенд для отрисовки графики (скажем, `qt`). В таких случаях нужно явно

потребовать, чтобы `matplotlib` использовал именно его. Посмотреть список доступных бэкендов можно с помощью

In [78]:

```
1 %matplotlib --list
```

```
Available matplotlib backends: ['tk', 'gtk', 'gtk3', 'wx', 'qt4', 'qt5', 'qt', 'osx', 'nbagg', 'notebook', 'agg', 'svg', 'pdf', 'ps', 'inline', 'ipympl', 'widget']
```

Как вы видите, `notebook` и `inline` это тоже варианты бэкенда.

%time, %timeit, %%time

Эти три команды измеряют, как долго выполняется код.

`%timeit` запускает команду несколько раз. По результатам вычисляется среднее и стандартное отклонение. Это уместно, если функция зависит от случайных битов. Для примера напомним функцию, которая генерирует случайное число и засыпает на одну секунду, если оно нечётное.

In [10]:

```
1 import time
2
3 def sleep_if_odd():
4     rvs = np.random.randint(low=0, high=int(1e9))
5     randbit = rvs % 2
6     if randbit % 2 == 1:
7         time.sleep(1)
```

In [9]:

```
1 %time sleep_if_odd()
```

```
CPU times: user 1.36 ms, sys: 423 µs, total: 1.78 ms
Wall time: 1 s
```

При использовании `%timeit` можно указать, сколько раз нужно запустить функцию. Синтаксис — `%timeit -n Дефолтное значение` — 7.

In [134]:

```
1 %timeit -n 10 sleep_if_odd()
```

```
487 ms ± 125 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

Видно, что только результаты `%timeit` отражают реальное поведение функции. Это стоит иметь в виду при тестировании кода.

%system, %%bash

Позволяет выполнять системные команды так, будто вы работаете в терминале.

In [81]:

```
1 %system date
```

Out[81]:

```
['Ср окт  9 21:08:35 MSK 2019']
```

Отметим, что для этого есть удобный альтернативный синтаксис:

In [84]:

```
1 !date
```

```
Ср окт  9 21:09:00 MSK 2019
```

С помощью команды `%%bash` можно вызвать многострочный скрипт на баше:

In [136]:

```
1 ▾ %%bash
2
3 ▾ for word in {'IPython', 'and', 'bash' 'are', 'friends!'}
4 do
5     echo $word
6 done
```

```
{IPython,
and,
bash
are,
friends!}
```

%debug

Мало кто знает, что в IPython (а следовательно и в jupyter) есть встроенный дебаггер.

Об этом есть замечательная [ветка сообщений в Twitter](https://twitter.com/radekosmulski/status/945739571735748609)

(<https://twitter.com/radekosmulski/status/945739571735748609>), мы кратко воспроизведём её здесь.

Напишем злободневную функцию и целенаправленно её сломаем:

In [125]:

```
1 from datetime import date, datetime, timedelta
2
3 deadline = datetime.combine(date.today(), datetime.max.time())
4 ▾ def submit_stats_homework(submission_time, deadline=deadline):
5     good_student = True
6 ▾     if submission_time > deadline:
7         good_student = False
8 ▾         raise ValueError(
9             "Вы сдали задание после дедлайна! 0 баллов! `_(ツ)_/`\n"
10            "Впредь будьте аккуратнее!"
11        )
```

In [129]:

```
1 submission_time = deadline + timedelta(seconds=1)
2 submit_stats_homework(submission_time)
```

```
-----
-----
ValueError                                Traceback (most recent call
last)
<ipython-input-129-30a256cb5a0d> in <module>
      1 submission_time = deadline + timedelta(seconds=1)
----> 2 submit_stats_homework(submission_time)

<ipython-input-125-ab85f7c267de> in submit_stats_homework(submission_t
ime, deadline)
      7         good_student = False
      8         raise ValueError(
----> 9             "Вы сдали задание после дедлайна! 0 баллов! ^\_
(ツ)_/^\n"
      9
      10         "Впредь будьте аккуратнее!"
      11     )

ValueError: Вы сдали задание после дедлайна! 0 баллов! ^\_ (ツ)_/^\n
Впредь будьте аккуратнее!
```

Вызов команды `%debug` вернёт вас к тому моменту исполнения, когда код упал с ошибкой. В консоли интерпретатора вы сможете посмотреть значение локальных переменных. Дебаггер становится незаменим, если функция работает долго или ошибку трудно воспроизвести.

In [130]:

```
1 %debug

> <ipython-input-125-ab85f7c267de>(9)submit_stats_homework()
7     good_student = False
8     raise ValueError(
----> 9         "Вы сдали задание после дедлайна! 0 баллов! ~\_(ツ)_/~\n"
10         "Впредь будьте аккуратнее!"
11     )
```

ipdb> help

Documented commands (type help <topic>):

```
=====
EOF      cl      disable  interact  next      psource   rv        unt
a        clear   display  j         p         q         s        until
alias    commands down     jump      pdef      quit      source   up
args     condition enable   l         pdoc      r         step     w
b        cont    exit     list      pfile     restart   tbreak   whati
s
break    continue  h        ll         pinfo     return    u        where
bt       d         help     longlist  pinfo2    retval    unalias
c        debug   ignore   n         pp         run       undisplay
```

Miscellaneous help topics:

```
=====
exec     pdb
```

```
ipdb> good_student
False
ipdb> submission_time > deadline
True
ipdb> exit
```

%%writefile, %pycat

Эти функции позволяют вам сохранить ячейку с кодом в питоновский файл и вывести его содержимое с подсветкой синтаксиса.

Это пригодится вам при работе с Google Colab весной (там нельзя менять .py -файлы, только перезаписывать).

In [148]:

```
1 ▾ %%writefile test.py
2   print("Hello, World!")
```

Writing test.py

In [146]:

```
1 %pycat test.py
```

In [147]:

```
1 !rm test.py
```

Работа на удалённом сервере

Если на вашем кластере не настроена точка входа, на которой уже запущен jupyter -сервер, это не беда.

Немного подправьте скрипт из ячейки ниже и сохраните его как jupyter_cluster.sh где-нибудь. Скорее всего, он заработает без дополнительных усилий с вашей стороны.

```
#!/usr/bin/bash
## get tunneling info
XDG_RUNTIME_DIR=""
ipnport=$(shuf -i8000-9999 -n1)
ipnip=$(hostname -i)
port=$1

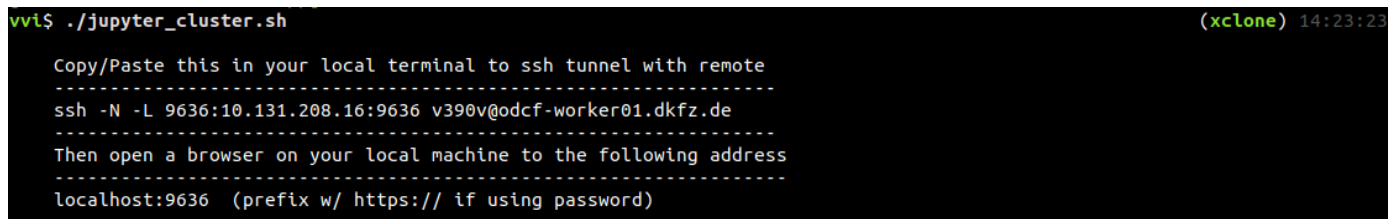
echo -e "
Copy/Paste this in your local terminal to ssh tunnel with remote
-----
ssh -N -L $ipnport:$ipnip:$ipnport $ТОЧКА_ВХОДА
-----

Then open a browser on your local machine to the following address
-----
localhost:$ipnport (prefix w/ https:// if using password)
-----
"

## start an ipcluster instance and launch jupyter server
/usr/bin/ssh -N -f -R localhost:$ipnport:localhost:$ipnport $ТОЧКА_ВХОДА
jupyter-notebook --no-browser --port=$ipnport --ip=$ipnip --notebook-dir=
'$ПАПКА_ДЛЯ_ВРЕМЕННЫХ_ФАЙЛОВ'
```

Сделайте файл исполняемым с помощью команды `chmod +x`, после этого его можно будет запускать как `./jupyter_cluster.sh`.

Давайте разберёмся, что же он делает.



```
vvi$ ./jupyter_cluster.sh (xclone) 14:23:23

Copy/Paste this in your local terminal to ssh tunnel with remote
-----
ssh -N -L 9636:10.131.208.16:9636 v390v@odcf-worker01.dkfz.de
-----
Then open a browser on your local machine to the following address
-----
localhost:9636 (prefix w/ https:// if using password)
-----
```

1. Он открывает ssh -тоннель на произвольном свободном порту сервера. Затем он выводит на экран команду, которая откроет тоннель с вашей стороны. Введите её в отдельном окошке терминала. ТОЧКА_ВХОДА имеет формат "ваш логин@адрес сервера". Например, [anonymous@calc.cod.phystech.edu \(mailto:anonymous@calc.cod.phystech.edu\)](mailto:anonymous@calc.cod.phystech.edu).

2. Затем он поднимает `jupyter` -сервер, временные файлы которого будут храниться в выбранной вами папке. Доступ к этому серверу будет осуществляться по тому же порту, к которому вы подключились на прошлом шаге.
3. Теперь вы можете открыть браузер, ввести в адресной `localhost` с тем номером порта, который вам показал скрипт, и работать в `jupyter` так же, как если бы он был запущен на вашем компьютере.

Полезные мелочи

В этом разделе мы поговорим о простых лайфхаках, которые сделают вашу работу с `jupyter` приятнее.

tqdm

`tqdm` (<https://github.com/tqdm/tqdm>) — та-ка-дум, что на арабском означает "прогресс" — прогресс-индикатор для Python, который показывает, сколько итераций уже сделано, сколько нужно и сколько осталось подождать. Не зная про него преступно, это незаменимый инструмент.

```
In [*]: from tqdm import trange, tqdm_notebook
        from time import sleep
        for i in tqdm_notebook(xrange(4), desc='1st loop'):
            for j in tqdm_notebook(xrange(100), desc='2nd loop', leave=False):
                sleep(0.01)
```

Центрирование графиков

Достаточно поместить эти строки в файл `~/.jupyter/custom/custom.css`.

```
.output_png {
    display: table-cell;
    text-align: center;
    vertical-align: middle;
}
```

После этого графики во всех ноутбуках будут выровнены по центру, что эстетично и часто удобно (особенно на больших мониторах).

Альтернативные цветовые схемы

Если вы хотите сменить цветовую схему `jupyter` -ноутбука (например, на привычную тёмную тему в духе `monokai`), то для этого вам нужно установить библиотеку `jupyterthemes`:

```
mamba install -c conda-forge jupyterthemes
```

и следовать инструкциям с [официальной страницы репозитория \(https://github.com/dunovank/jupyter-themes\)](https://github.com/dunovank/jupyter-themes).

The screenshot shows a Jupyter Notebook interface in a web browser. The notebook is titled "Jupyter Themes Demo (autosaved)". The toolbar includes standard Jupyter actions like File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. The code area contains three sections:

Jupyter Themes

For list of available themes

```
In [67]: !jt -l
```

Available Themes:

```
chesterish
grade3
gruvboxd
gruvboxl
monokai
oceans16
onedork
solarizedd
solarizedl
```

Selecting a particular theme

```
In [105]: !jt -t oceans16
```

Reverting to original Theme

```
In [ ]: !jt -r
```

Заключение

В этом ноутбке рассмотрены базовые возможности `jupyter`, которые, надеюсь, помогут вам в решении повседневных задач.

Тем не менее, это далеко не всё! Планируется рассмотреть такие темы как:

- совместная работа над `.ipynb`-файлами с помощью `jupyter hub` (полезно для хакатонов);
- создание презентаций в `jupyter`;
- работа с CPython, R и C++ в `jupyter`;
- аналоги `jupyter-notebook`: `jupyter-lab`, `rmarkdown`, `observable notebooks`;