



Язык Python. История, Особенности и Возможности

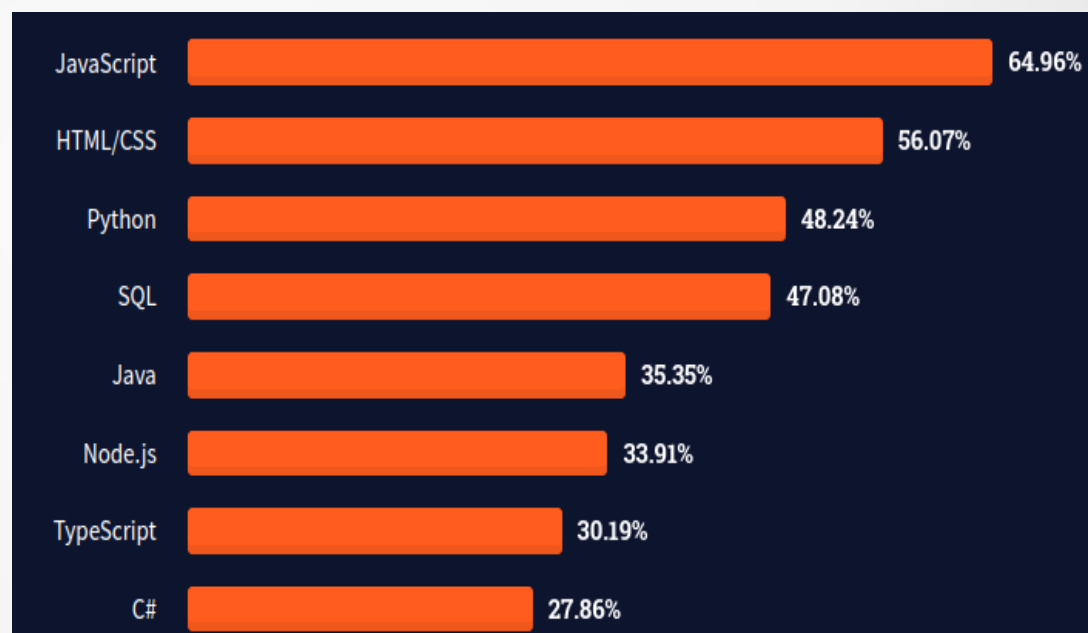
Евгений Борисов

Python. Особенности и возможности

Рейтинг языков программирования TIOBE

Nov 2022	Programming Language	
1		Python
2		C
3		Java
4		C++
5		C#

Рейтинг технологий программирования
StackOverflow



Python. Особенности и возможности

Проект с историей, существует уже более 30 лет.

Поддержка современных технологий в библиотеках.

Открытое сообщество — доступен всем, над разработкой работают энтузиасты со всего мира.

Универсален — подходит почти для любых решений в области программирования.

Мультиплатформенный — есть реализации почти для всех операционных систем и аппаратных платформ.

Python. Особенности и возможности

Язык программирования общего назначения,

Высокоуровневый,

Императивный, объектно-ориентированный,

Строгая динамическая типизация

Python. Особенности и возможности

Python — высокоуровневый язык программирования.

Высокоуровневый язык программирования — оптимизирован для удобства использования, применяются абстракции — структуры данных, набор вспомогательных функций и т.п.

Низкоуровневый язык — оптимизирован для эффективности выполнения, близок к машинному коду и его конструкциям (Assembler).

Python. Особенности и возможности

Python — императивный язык программирования.

Императивный язык - программа это строго упорядоченный список команд для выполнения.

Декларативный язык - программа это описания результата, который мы хотим получить (SQL)

Python. Особенности и возможности

Python - объектно-ориентированный язык программирования,
поддерживает процедурный, структурный и функциональный стиль.

Парадигмы (стили) программирования:

Процедурная — программа строго упорядоченный список команд (Assembler, Shell)

Структурная — программа набор подпрограмм, выполняемый в определённом порядке.

Объектно-ориентированная — программа как набор деталей встроенных друг в друга образующих вместе единый механизм.

Функциональная — программа как суперпозиция математических функций.

Python. Особенности и возможности

Python — строго типизированный язык программирования с возможностью динамической типизации.

Строго типизированный язык - определён ограниченный список типов данных

Динамическая типизация - в процессе выполнения программы переменная может связываться с данными разных типов, объявляем переменную не указываем явно, какой тип данных в ней будет содержаться.

Статическая типизация - тип переменной объявляется явно и в процессе выполнения программы он не меняется.

Python. Особенности и возможности

Существуют реализации Python как интерпретатора, так и компилятора.



Интерпретация — программа оптимизируется и выполняется интерпретатором (специальной виртуальной машиной).

- может выполняться медленно;
- + независимость от платформы, меньший размер;



Компиляция — программа преобразуется в машинный код (исполняемый файл), который выполняется аппаратной частью непосредственно.



- ограниченная переносимость, ограничения на инструментарий языка;
- + можно добиться оптимального использования вычислительных ресурсов;



Python. Особенности и возможности

Python имеет очень много разнообразных библиотек и фреймворков

The Python Package Index (PyPI) is a repository of software
for the Python programming language.

<http://pypi.org>

Python. Особенности и возможности

примеры приложений реализованных на Python

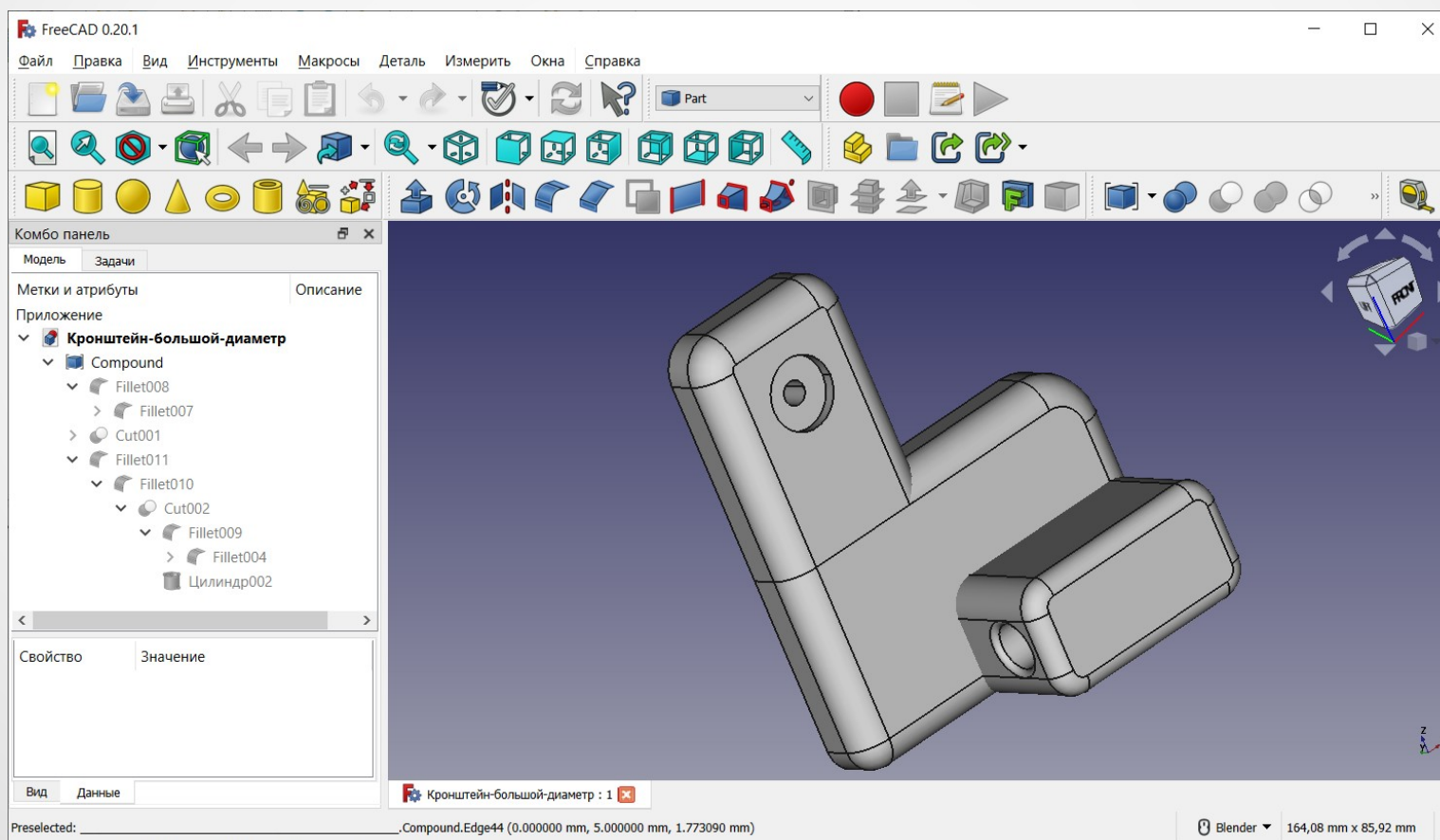
World of Tanks - <http://tanki.su>



Python. Особенности и возможности

примеры приложений реализованных на Python

FreeCad - <http://www.freecadweb.org>



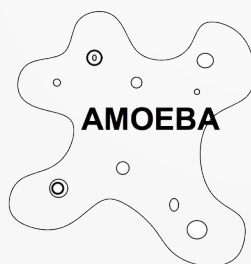
Python. Особенности и возможности



автор первой версии Python
- голландский программист Гвидо ван Россум

центр математики и информатики в Нидерландах,

изначально планировался как язык сценариев для системы Амоеба,
начало проекта в декабре 1989 года



Амоеба — открытая микроядерная
распределённая операционная система,
разработанная группой во главе с Эндрю
Таненбаумом в Амстердамском свободном
университете. <https://www.cs.vu.nl/pub/amoeba/>



Amoeba



Developed at:

- Vrije Universiteit (Amsterdam)
- (Free University)

In cooperation with:

- Centrum voor Wiskunde en Informatica (Amsterdam)
- (Center for Mathematics and Computer Science)
- Research began in 1980

Python. Особенности и возможности

20 февраля 1991 года через сеть Usenet был опубликован код языка Python .

Так появилась первая версия языка с номером 0.9.0

В языке уже присутствовали:

- основные типы данных (list, dict, str),
- поддержка модулей (пакеты подпрограмм),
- классы с наследованием,
- обработка исключений,

В январе 1994 года вышла Python 1.0

Python. Особенности и возможности



Гвидо назвал свой язык в честь комедийного телесериала «Летающий цирк Монти Пайтона»



старый логотип (просуществовал до 2006 года)



Why Python?

Python. Особенности и возможности

Правовые вопросы Python

20 февраля 1991 года был впервые опубликован код языка Python версии 0.9.0.

29 июня 1994 года вышла статья Майкла Маклей из Национального института стандартов и технологий США (NIST) «Если бы Гвидо сбил автобус?»

<https://legacy.python.org/search/hypermail/python-1994q2/1040.html>

Публикация затронула проблему зависимости Python-сообщества от решений Гвидо

в 1995 году была создана Python Software Foundation — некоммерческая организация, которая должна была отвечать за защиту и развитие языка Python.

Контроль за соблюдением порядка осуществляет «совет руководителей», регулярно переизбирается и состоит из пяти человек,

Гвидо ван Россум получил шуточный титул «великодушного пожизненного диктатора» (BDFL, Benevolent Dictator For Life).

в 2018 году Гвидо отказался от титула BDFL и сделал язык Python полностью независимой технологией.

Python. Особенности и возможности

3 декабря 2008 года выходит Python 3.0, код 3.x и 2.x версии совместим частично

до 2020 2.x и 3.x ветки развивались параллельно

с конца 2020 поддержка ветки 2.x была официально завершена

Python. Особенности и возможности

Python Enhancement Proposals (PEPs)
<https://peps.python.org>

Предложения по улучшению Python - официальная документация языка.

Позиции из списка PEPs открыто обсуждаются сообществом Python.

PEP 8 Style Guide for Python Code / Руководство по оформлению кода.

PEP 13 Python Language Governance / Список руководителей проекта.

Python. Особенности и возможности

PEP20 - 19 правил по улучшению языка Питон от Тима Петерса,

1. Красивое лучше уродливого.
2. Явное лучше неявного.
3. Простое лучше сложного.
4. Сложное лучше запутанного.
5. Развёрнутое лучше вложенного.
6. Разреженное лучше плотного.
7. Читаемость имеет значение.
8. Особые случаи не настолько особые, чтобы нарушать правила.
9. При этом практичность важнее безупречности.
10. Ошибки не должны замалчиваться.
11. Если не замалчиваются явно.
12. Встретив двусмысленность, отбрось искушение угадать.
13. Должен существовать один - и, желательно, только один – очевидный способ сделать что-то.
14. Хотя этот способ поначалу может быть и не очевиден, если вы не голландец.
15. Сейчас лучше, чем никогда.
16. Хотя никогда часто лучше, чем *прямо* сейчас.
17. Если реализацию сложно объяснить – идея точно плоха.
18. Если реализацию легко объяснить – возможно, идея хороша.
19. Пространства имен – отличная штука! Будем использовать их чаще!

Python: дистрибутивы



CPython

Anaconda (Miniconda)



Python: менеджер пакетов pip



CPython

```
# pip search pep8
```

```
# pip install autopep8
```

```
# pip list
```

```
# pip uninstall autopep8
```

Python: утилиты

показывает места нарушения стиля
pep8 --first main.py

определяет и исправляет нарушения стиля
autopep8 ./ --recursive --in-place -a

форматирует комментарии
docformatter --in-place example.py

универсальная утилита приведения кода к PEP
pyformat

Python: virtual environments

проблема: пакеты определённых версий могут быть несовместимы между собой

решение: виртуальные python-среды

позволяет работать с несколькими версиями python

держат одновременно несколько наборов пакетов разных версий

venv - creation of virtual environments <https://docs.python.org/3/library/venv.html>

```
# sudo mkdir /opt/venv
```

```
# sudo chown -R USER /opt/venv
```

```
# cd /opt/venv
```

```
# python3 -m venv /opt/venv/jupyter_1
```

```
# source /opt/venv/jupyter_1/bin/activate
```

```
# pip3 install numpy ....
```

Python: IDE

IDLE

iPython / Jupyter

PyCharm

Visual Studio Code

Eclipse + PyDev

Vim

Apache Zeppelin



IP[y]:



Python: Jupyter

<https://jupyter.org>

The image is a collage of several Jupyter Notebook windows, illustrating different data analysis workflows. The top window, titled 'In Depth: Linear Regression', contains text explaining the basics of linear regression models. Below it, a 'Simple' notebook shows a scatter plot of data points and a linear regression line. The bottom row features three smaller notebooks: 'Julia' showing a scatter plot, 'python notebook' showing the Lorenz attractor and its equations, and 'R' showing a scatter plot of the Iris dataset. The right side of the collage shows a 'Seattle Weather: 2012-2015' notebook with a scatter plot of temperature over time and a bar chart of precipitation. The bottom right corner shows a table of Iris dataset data.

In Depth: Linear Regression

Just as naive Bayes (discussed earlier in [In Depth: Naive Bayes Classification](#)) is a good starting point for classification tasks, linear regression models are a good starting point for regression tasks. Such models are popular because they can be fit very quickly, and are very interpretable. You are probably familiar with the simplest form of a linear regression model (i.e., fitting a straight line to data) but such models can be extended to model more complicated data behavior.

In this section we will start with a quick intuitive walk-through of the mathematics behind this well-known problem, before seeing how before moving on to see how linear models can be generalized to account for more complicated patterns in data.

We begin with a quick walk-through of the mathematics behind this well-known problem, before seeing how before moving on to see how linear models can be generalized to account for more complicated patterns in data.

Simple

We will start with a simple example where a linear regression model is used to predict the value of a variable.

```
[1]: rng = np.random.RandomState(0)
x = 10 * rng.rand(100)
y = 2 * x + rng.randn(100)
plt.scatter(x, y)
```

Julia

```
[1]: using Datasets, Seaborn
plot(datasets["iris"], x="Sepal.Length", y="Petal.Length")
```

python notebook

```
[1]: from lorenz import solve_lorenz
x = solve_lorenz(0, 0, 50)
y = solve_lorenz(0, 0, 50)
z = solve_lorenz(0, 0, 50)
plot(x, y, z)
```

R

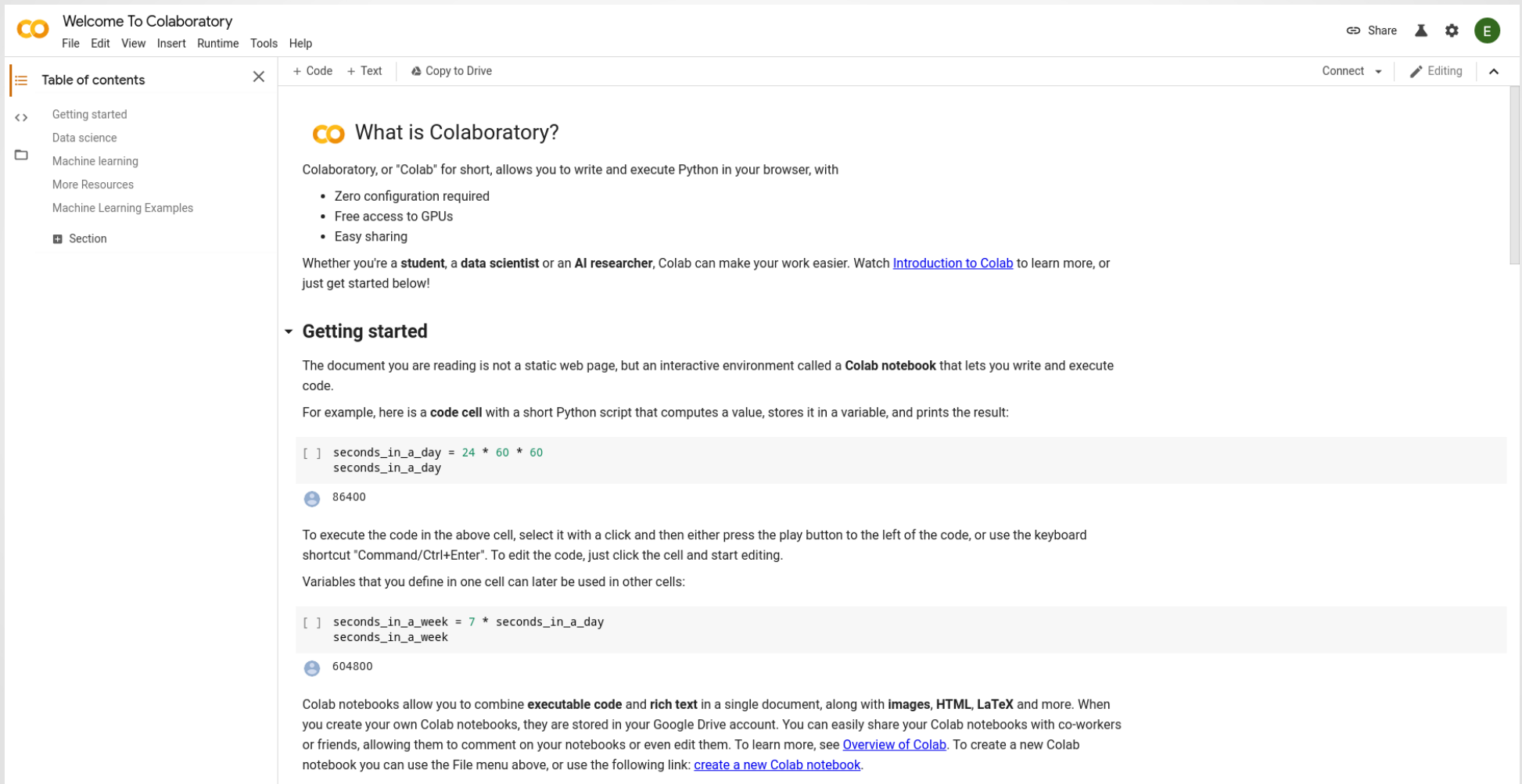
```
[1]: ggplot(data=iris, aes(x=Sepal.Length, y=Petal.Length))
geom_point()
```

Sepal.Length	Sepal.Width	Petal.Length
5.1	3.5	1.4
4.9	3.0	1.4

Python: Google Colab

<https://colab.research.google.com/>

<https://habr.com/ru/post/348058/>



The screenshot shows the Google Colaboratory (Colab) interface. At the top, there's a header with the Colab logo and the text "Welcome To Colaboratory". Below this is a menu bar with options: File, Edit, View, Insert, Runtime, Tools, and Help. On the right side of the header, there are icons for Share, a user profile, settings, and a green circle with a white 'E'. Below the header, there's a sidebar on the left with a "Table of contents" section. The main content area is titled "What is Colaboratory?" and contains the following text:

Colaboratory, or "Colab" for short, allows you to write and execute Python in your browser, with

- Zero configuration required
- Free access to GPUs
- Easy sharing

Whether you're a **student**, a **data scientist** or an **AI researcher**, Colab can make your work easier. Watch [Introduction to Colab](#) to learn more, or just get started below!

Getting started

The document you are reading is not a static web page, but an interactive environment called a **Colab notebook** that lets you write and execute code.

For example, here is a **code cell** with a short Python script that computes a value, stores it in a variable, and prints the result:

```
[ ] seconds_in_a_day = 24 * 60 * 60
seconds_in_a_day
```

86400

To execute the code in the above cell, select it with a click and then either press the play button to the left of the code, or use the keyboard shortcut "Command/Ctrl+Enter". To edit the code, just click the cell and start editing.

Variables that you define in one cell can later be used in other cells:

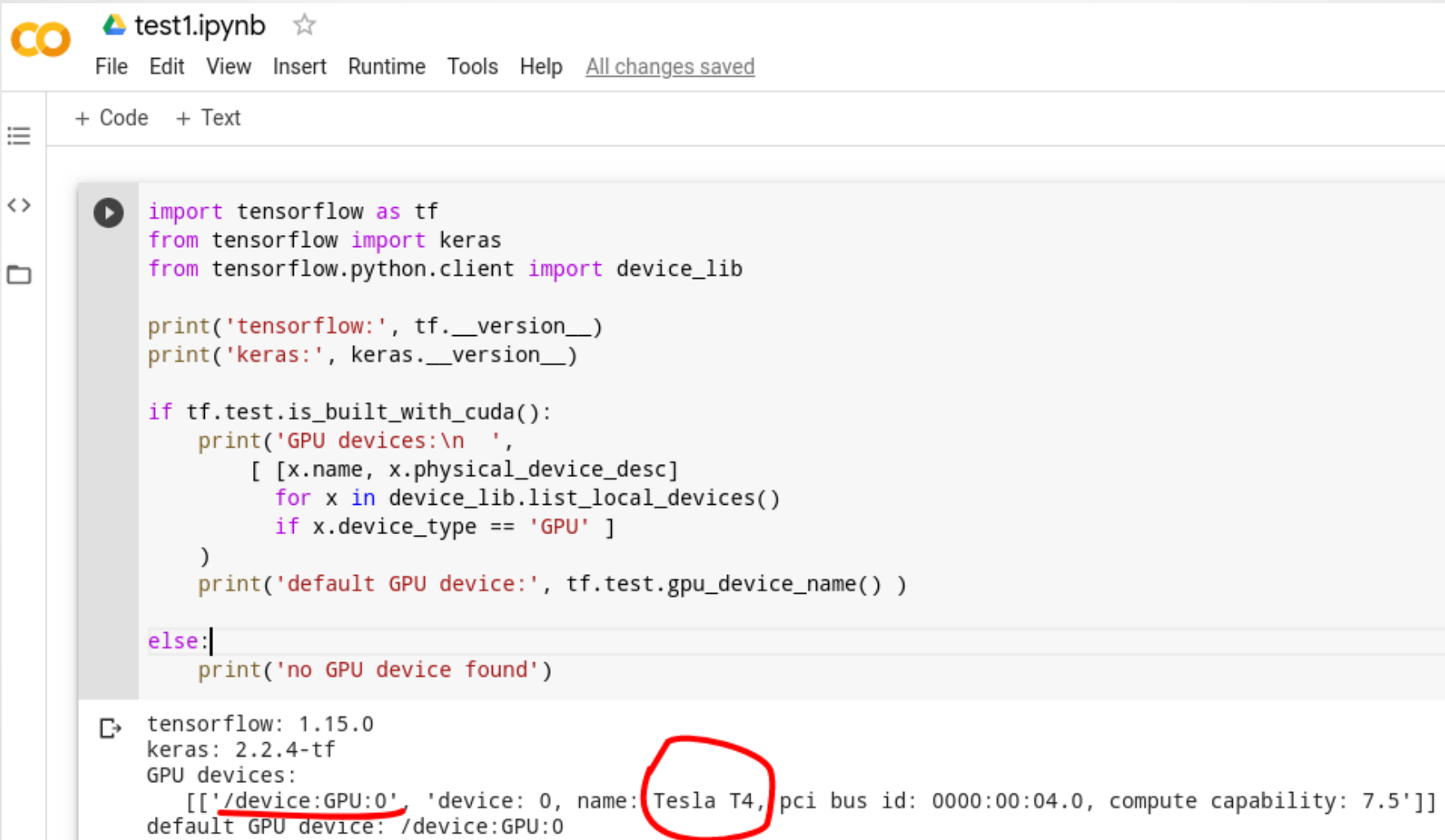
```
[ ] seconds_in_a_week = 7 * seconds_in_a_day
seconds_in_a_week
```

604800

Colab notebooks allow you to combine **executable code** and **rich text** in a single document, along with **images**, **HTML**, **LaTeX** and more. When you create your own Colab notebooks, they are stored in your Google Drive account. You can easily share your Colab notebooks with co-workers or friends, allowing them to comment on your notebooks or even edit them. To learn more, see [Overview of Colab](#). To create a new Colab notebook you can use the File menu above, or use the following link: [create a new Colab notebook](#).

Python: Google Colab

<https://colab.research.google.com/>



The image shows a Google Colab notebook interface. At the top, there's a green header with the text "Python: Google Colab". Below it, the URL "https://colab.research.google.com/" is displayed. The notebook itself has a title "test1.ipynb" and a star icon. The menu bar includes "File", "Edit", "View", "Insert", "Runtime", "Tools", and "Help", with a status "All changes saved". On the left, there are icons for a file explorer, a code editor, and a text editor. The code editor shows a code cell with the following Python code:

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.python.client import device_lib

print('tensorflow:', tf.__version__)
print('keras:', keras.__version__)

if tf.test.is_built_with_cuda():
    print('GPU devices:\n ',
          [ [x.name, x.physical_device_desc]
            for x in device_lib.list_local_devices()
            if x.device_type == 'GPU' ]
          )
    print('default GPU device:', tf.test.gpu_device_name() )
else:
    print('no GPU device found')
```

The output of the code cell is displayed below the code:

```
tensorflow: 1.15.0
keras: 2.2.4-tf
GPU devices:
[['/device:GPU:0', 'device: 0, name: Tesla T4, pci bus id: 0000:00:04.0, compute capability: 7.5']]
default GPU device: /device:GPU:0
```

In the output, the text "/device:GPU:0" is underlined with a red line, and the text "Tesla T4" is circled with a red line.

Python: Yandex DataSphere

<https://cloud.yandex.ru/blog/posts/2020/05/datasphere>

The screenshot displays the Yandex DataSphere IDE interface. On the left, a file explorer shows a directory with files: `catboost_info`, `train_data.npy`, `train_labels.npy`, and `yadisk.token`. The main editor window, titled `test.ml`, contains a Python script for training a CatBoost classifier. The script includes imports for `CatBoostClassifier` and `numpy`, followed by loading the training data and labels, creating the classifier with specific parameters, and fitting the model. The output shows the learning rate set to `0.003512` and a progress log for 30 iterations, detailing learning rates, total times, and remaining times.

```
[1]: from catboost import CatBoostClassifier
import numpy as np

[3]: train_data = np.load('train_data.npy')
train_labels = np.load('train_labels.npy')

[5]: model = CatBoostClassifier(iterations=1000,
                                task_type="CPU",
                                devices='0:1')

[6]: model.fit(train_data, train_labels, verbose=True)

Learning rate set to 0.003512
0:   learn: 0.6906569   total: 49.8ms   remaining: 49.7s
1:   learn: 0.6881785   total: 50.1ms   remaining: 25s
2:   learn: 0.6871994   total: 50.2ms   remaining: 16.7s
3:   learn: 0.6864504   total: 50.3ms   remaining: 12.5s
4:   learn: 0.6839919   total: 50.4ms   remaining: 10s
5:   learn: 0.6815451   total: 50.5ms   remaining: 8.37s
6:   learn: 0.6791099   total: 50.6ms   remaining: 7.18s
7:   learn: 0.6783759   total: 50.7ms   remaining: 6.29s
8:   learn: 0.6776453   total: 50.8ms   remaining: 5.59s
9:   learn: 0.6757266   total: 50.9ms   remaining: 5.04s
10:  learn: 0.6733133   total: 51ms     remaining: 4.58s
11:  learn: 0.6723679   total: 51.1ms   remaining: 4.21s
12:  learn: 0.6704690   total: 51.2ms   remaining: 3.89s
13:  learn: 0.6685772   total: 51.3ms   remaining: 3.61s
14:  learn: 0.6661971   total: 51.4ms   remaining: 3.38s
15:  learn: 0.6633391   total: 51.5ms   remaining: 3.17s
16:  learn: 0.6609833   total: 51.6ms   remaining: 2.98s
17:  learn: 0.6586417   total: 51.7ms   remaining: 2.82s
18:  learn: 0.6567942   total: 51.8ms   remaining: 2.68s
19:  learn: 0.6549535   total: 51.9ms   remaining: 2.54s
20:  learn: 0.6521563   total: 52ms     remaining: 2.42s
21:  learn: 0.6503330   total: 52.1ms   remaining: 2.32s
22:  learn: 0.6494321   total: 52.2ms   remaining: 2.22s
23:  learn: 0.6471437   total: 52.3ms   remaining: 2.13s
24:  learn: 0.6464625   total: 52.4ms   remaining: 2.04s
25:  learn: 0.6446604   total: 52.5ms   remaining: 1.97s
26:  learn: 0.6428651   total: 52.6ms   remaining: 1.9s
27:  learn: 0.6410765   total: 52.7ms   remaining: 1.83s
28:  learn: 0.6404061   total: 52.8ms   remaining: 1.77s
29:  learn: 0.6381544   total: 52.9ms   remaining: 1.71s
30:  learn: 0.6359178   total: 53ms     remaining: 1.66s
```

<https://habr.com/ru/article/565086/>

Python

Борисов Е.С. Методы машинного обучения. 2024
https://github.com/mechanoid5/ml_lectorium_2024_I

<https://www.python.org>

<https://jupyter.org>

<https://docs.python.org/3/library/venv.html>