Universitat de Barcelona

Tutorial COLAB

Dr. Simone Balocco

Mathematics and Computer Science dept.

University of Barcelona

Universitat de Barcelona

Contents

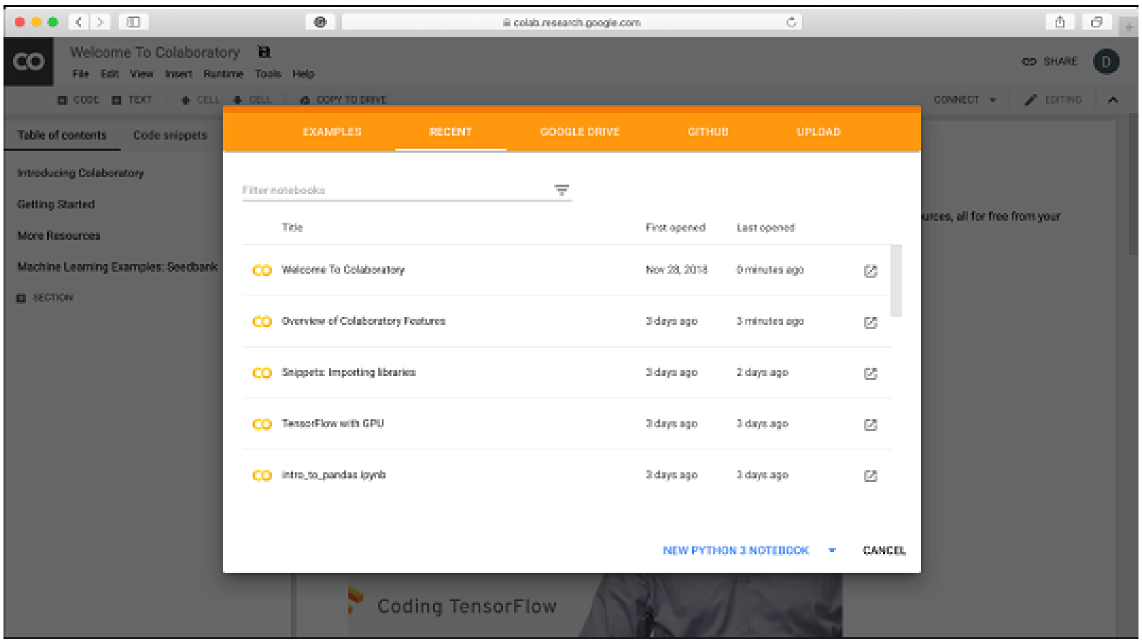
Universitat de Barcelona

1.Google Colab : your first Colab notebook

In this chapter, you will create and run your first trivial notebook. Follow the steps given to you whenever necessary.

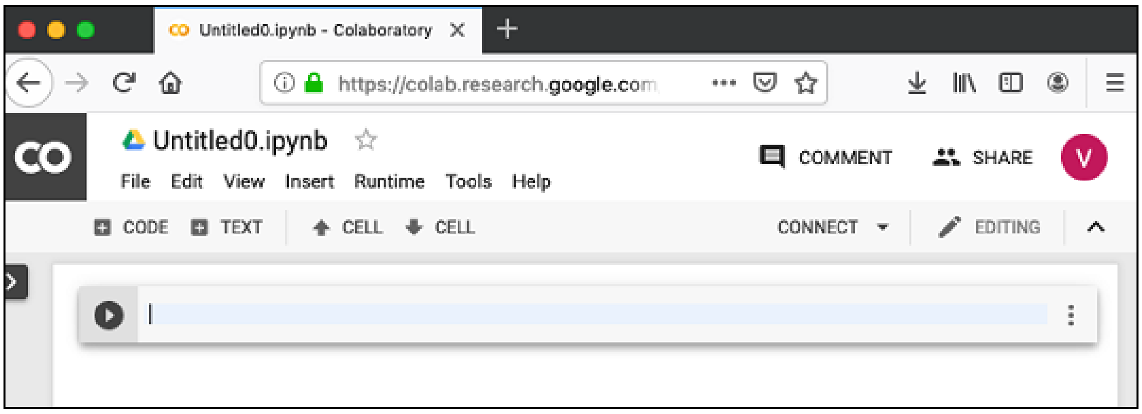
Note - Since Colab implicitly uses Google Drive to store your notebooks, make sure you're signed in to your Google Drive account before continuing.

Step 1 - Open the following URL in your browser - https://colab.research.google.com
Your browser will display the following screen (assuming you are logged into your Google Drive):



Title	First opened	Last opened
Welcome To Colaboratory	Nov 28, 2018	0 minutes ago
Overview of Colaboratory Features	3 days ago	3 minutes ago
Snippets: Importing libraries	3 days ago	2 days ago
TensorFlow with GPU	3 days ago	3 days ago
intro_to_pandas.ipynb	3 days ago	3 days ago

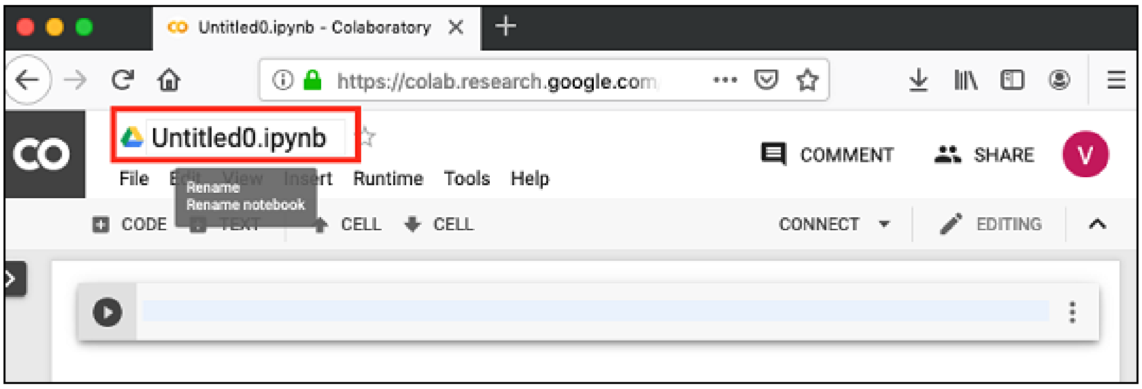
Step 2 - Click on the **NEW PYTHON 3 NOTEBOOK** link at the bottom of the screen. A new laptop will open as shown in the following screen.


Untitled0.ipynb - Colaboratory
https://colab.research.google.com
Untitled0.ipynb
File Edit View Insert Runtime Tools Help
COMMENT
SHARE
CODE TEXT CELL CELL
CONNECT
EDITING

As you may have noticed, the laptop interface is quite similar to the one provided in Jupyter . There is a code window where you need to enter your Python code.

1.1. Setting the notebook name

By default, the notebook uses the UntitledXX.ipynb naming convention. To rename the notebook, click on this name and type the desired name in the edit box as shown here -



We will call this notebook as **MyFirstColabNotebook** . So type this name in the edit box and press ENTER. The notebook will acquire the name you have given it now.

1.2. Entering code

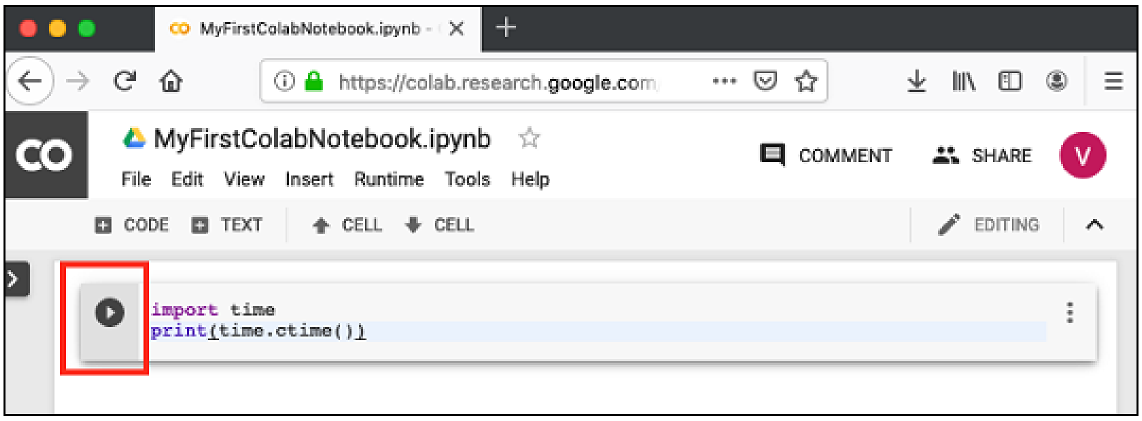
You will now enter some trivial Python code into the code window and run it.

Enter the following two Python statements in the code window:

```
import time
print ( time.ctime ())
```

1.3. running code

To run the code, click the arrow on the left side of the code window.

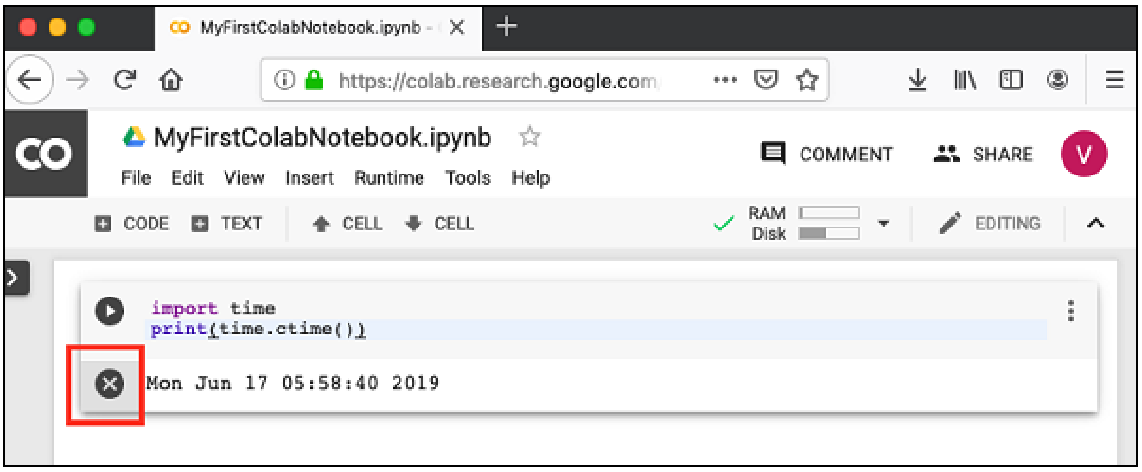


```
import time
print(time.ctime())
```

After a while, you will see the output below the code window, as shown here:

```
Mon Jun 17 05:58:40 2019
```

You can clear the output at any time by clicking the icon on the left side of the output screen.



```
import time
print(time.ctime())
```

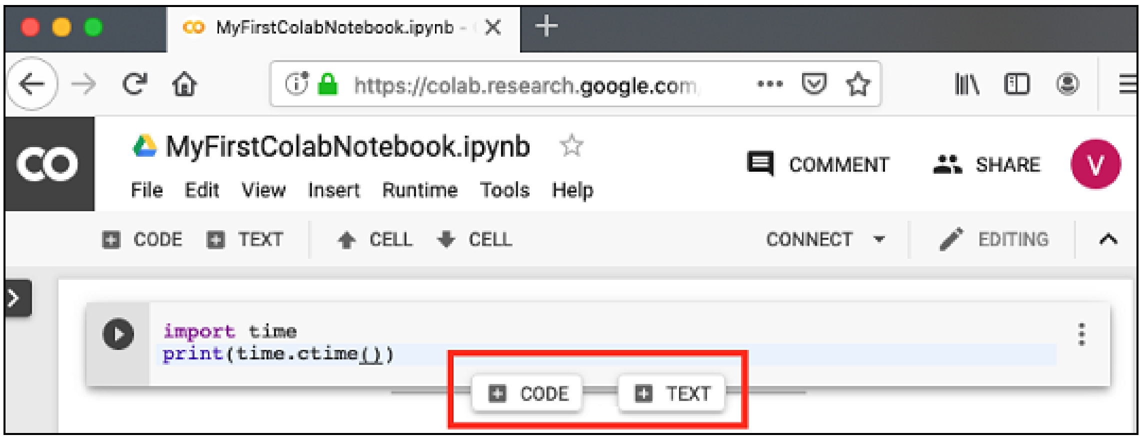
```
Mon Jun 17 05:58:40 2019
```

1.4. Add code cells

To add more code to your laptop, select the following **menu** options -

```
Insert / Code Cell
```

Alternatively, just hover your mouse in the bottom center of the Code cell. When the **CODE** and **TEXT buttons** appear , click CODE to add a new cell. This is shown in the screenshot below:


MyFirstColabNotebook.ipynb
https://colab.research.google.com
MyFirstColabNotebook.ipynb
File Edit View Insert Runtime Tools Help
COMMENT
SHARE
CODE TEXT CELL CELL
CONNECT
EDITING
CODE
TEXT

```
import time
print(time.ctime())
```

A new code cell will be added below the current cell. Add the following two declarations in the newly created code window:

```
time.sleep (5)
print( time.ctime ())
```

Now if you execute this cell, you will see the following output:

```
Mon Jun 17 04:50:27 2019
```

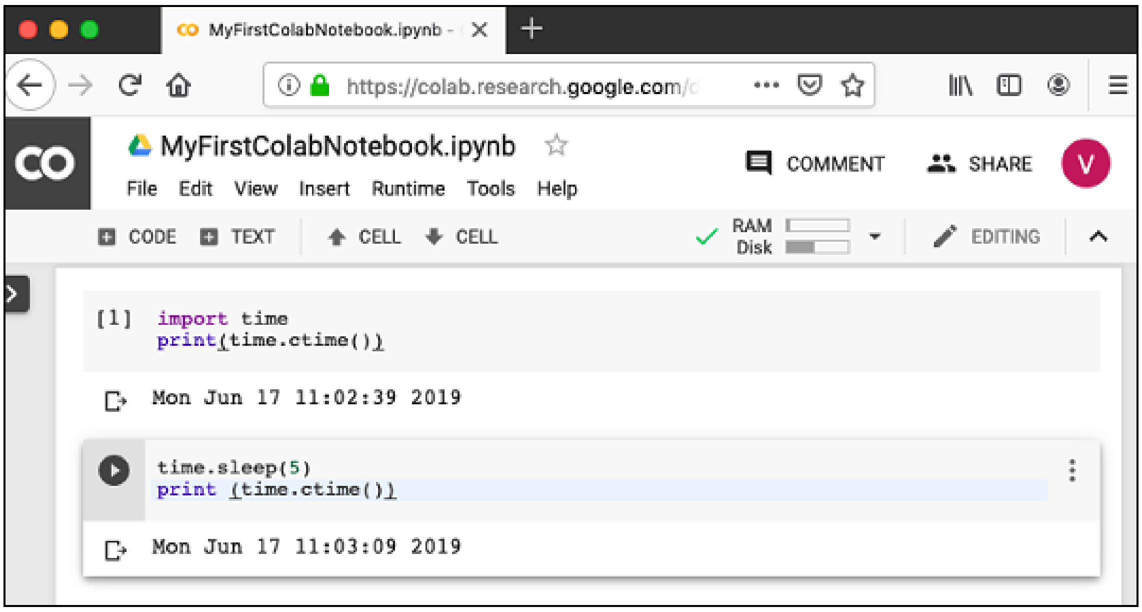
Certainly the time difference between the two time strings is not 5 seconds. This is obvious as it took a while to insert the new code. Colab allows you to run all your code inside your laptop without interruption.

1.5. run all

To run all the code on your laptop without interruption, run the following menu options:

```
Runtime / Reset and run all…
```

It will give you the output as shown below:


MyFirstColabNotebook.ipynb
https://colab.research.google.com/
MyFirstColabNotebook.ipynb
File Edit View Insert Runtime Tools Help
COMMENT SHARE
CODE TEXT CELL CELL
RAM Disk
EDITING

```
[1] import time
    print(time.ctime())

Mon Jun 17 11:02:39 2019

time.sleep(5)
print (time.ctime())

Mon Jun 17 11:03:09 2019
```

Note that the time difference between the two outputs is now exactly 5 seconds.

The above action can also be started by executing the following two menu options:

```
Runtime / Restart runtime…
```

either

```
Runtime / Restart all runtimes…
```

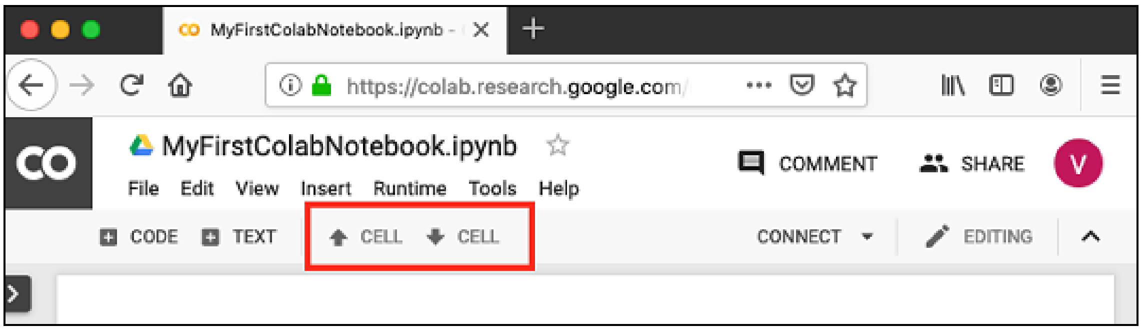
Followed by

```
Runtime / Run all
```

Study the different menu options in the **Runtime** menu to become familiar with the various options available to run the notebook.

1.6. Change cell order

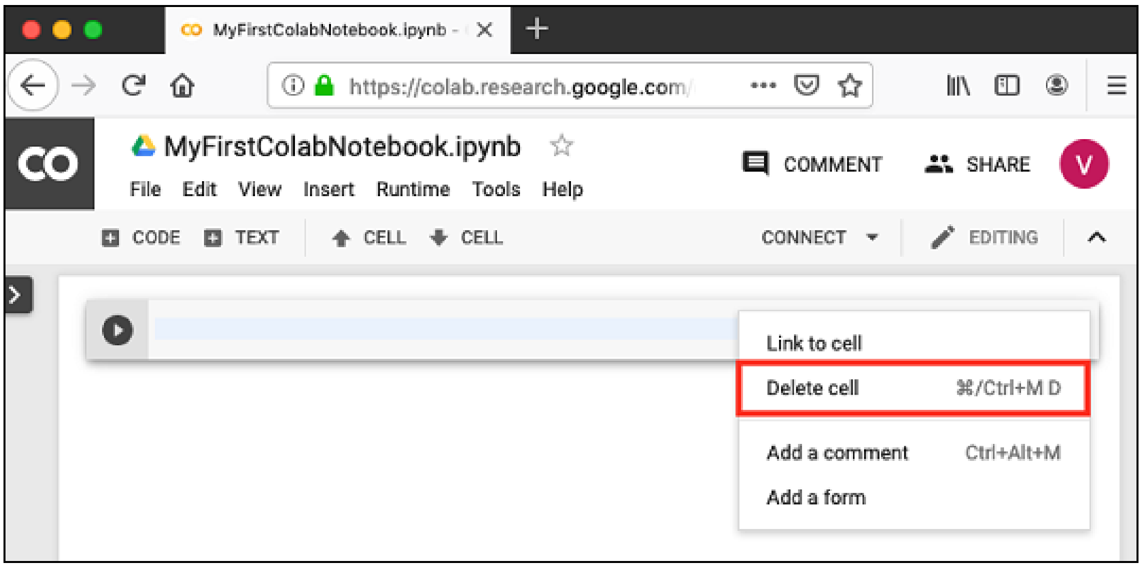
When your notebook contains a large number of code cells, you may run into situations where you would like to change the execution order of these cells. You can do this by selecting the cell you want to move and clicking the **UP CELL** or **DOWN CELL** buttons shown in the following screenshot:


MyFirstColabNotebook.ipynb
https://colab.research.google.com/
File Edit View Insert Runtime Tools Help
COMMENT
SHARE
CODE
TEXT
CELL
CELL
CONNECT
EDITING

You can click the buttons multiple times to move the cell more than one position.

1.7. deleting cell

During the development of your project, you may have entered some now unwanted cells in your notebook. You can easily remove these cells from your project with a single click. Click the vertical dots icon in the top right corner of your code cell.



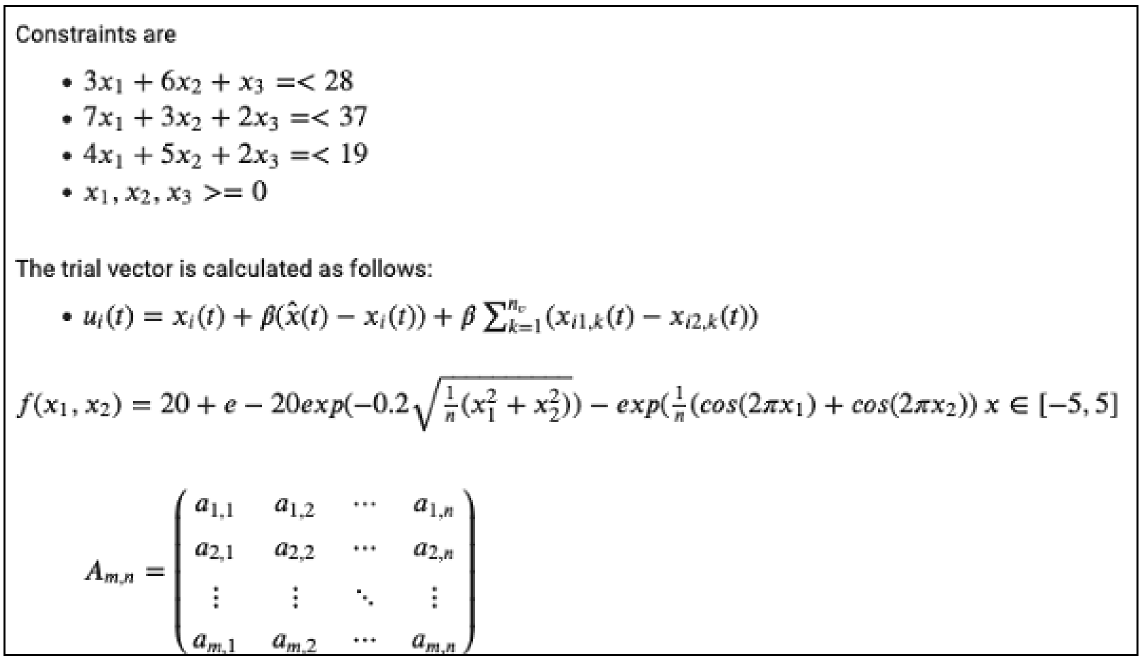
Click on the **Delete cell** option and the current cell will be deleted.

Now, since you've learned how to run a trivial notebook, let's explore the other capabilities of Colab .

2.Google Colab - documenting your code

Since the code cell supports full Python syntax, you can use Python **comments** in the code window to describe your code. However, many times more than a simple text-based commentary is needed to illustrate ML algorithms. ML uses math a lot, and to explain those terms and equations to your readers, you need an editor that supports LaTex , a language for mathematical representations. Colab offers **Text Cells** for this purpose.

The following screenshot shows a text cell that contains few math equations that are commonly used in ML:



Constraints are

- $3x_1 + 6x_2 + x_3 =< 28$
- $7x_1 + 3x_2 + 2x_3 =< 37$
- $4x_1 + 5x_2 + 2x_3 =< 19$
- $x_1, x_2, x_3 >= 0$

The trial vector is calculated as follows:

- $u_i(t) = x_i(t) + \beta(\hat{x}(t) - x_i(t)) + \beta \sum_{k=1}^{n_v}(x_{i1,k}(t) - x_{i2,k}(t))$

$$f(x_1, x_2) = 20 + e - 20exp(-0.2\sqrt{\frac{1}{n}(x_1^2 + x_2^2)}) - exp(\frac{1}{n}(cos(2\pi x_1) + cos(2\pi x_2)) \; x \in [-5, 5]$$

$$A_{m,n} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix}$$

As we go through this chapter, we'll look at the code to generate the above output.

Text cells are formatted using **markdown** - a simple markup language. Let's now see how to add text cells to your notebook and add text to it that contains mathematical equations.

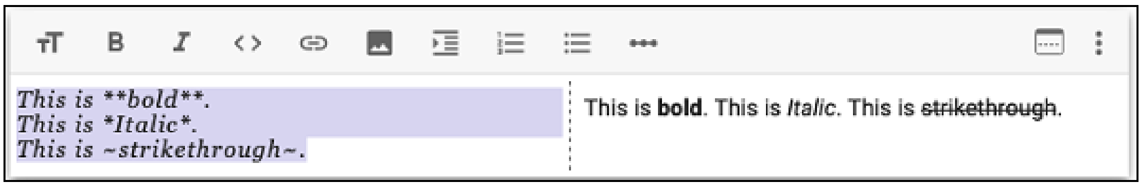
2.1. examples of rebates

Let's look at some examples of markup language syntax to demonstrate its capabilities.

Type the following text in the Text cell.

```
This is **bold**.
This is *italic*.
Este is ~ strikethrough ~.
```

The output of the above commands is rendered on the right side of the cell as shown here.



```
This is **bold**.
This is *Italic*.
This is ~strikethrough~.
```

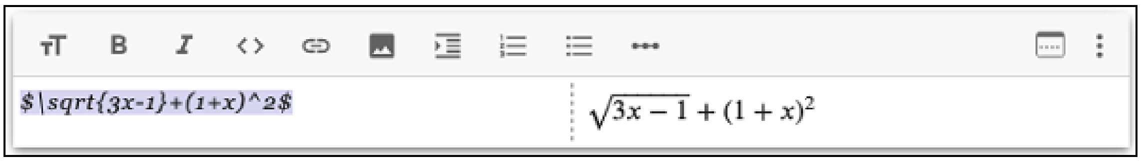
This is **bold**. This is *Italic*. This is ~~strikethrough~~.

2.2. Mathematical equations

Add a **Text Cell** to your notebook and enter the following markdown syntax in the text window:

```
$\ sqrt {3x- 1}+( 1+x)^2$
```

You will see the immediate rendering of the markdown code in the panel on the right side of the text cell. This is shown in the screenshot below:



```
$\sqrt{3x-1}+(1+x)^2$
```

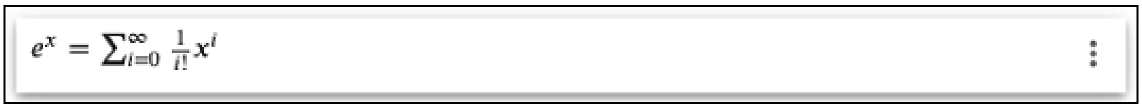
$\sqrt{3x-1}+(1+x)^2$

Hit **Enter** and the discount code disappears from the text cell and only the rendered output is displayed .

Let's try another more complicated equation as shown here:

```
$ e^x = \sum { i = 0}^\ infty \frac{1}{i!} x^i $
```

rendered output is shown here for your quick reference.



$e^x = \sum_{i=0}^\infty \frac{1}{i!} x^i$

2.3. Code for sample equations

Here is the code for the sample equations shown in a screenshot above:

```
Constraints are
- $3x_1 + 6x_2 + x_3 =< 28$
- $7x_1 + 3x_2 + 2x_3 =< 37$
- $4x_1 + 5x_2 + 2x_3 =< 19$
- $x_ 1,x _2,x_3 >=0 $

The trial vector is calculated as follows:
```

```
- $ u_i (t) = x_i (t) + \beta(\hat{x}(t) - x_i (t)) + \beta \sum k =
1}^{ n_v }(x_{i1,k }(t) - x_{i2,k}(t))$
$ f( x_1, x_2) = 20 + e - 20exp(-0.2 \ sqrt {\ frac {1}{n} (x_1^2 +
x_2^2)}) - exp (\ frac {1}{n}( cos(2\pi x_1) + cos(2\pi x_2))$

$x ∈ [-5, 5]$
>$A_{ m,n } =
\begin{ pmatrix }
a 1,1 > a 1,2 > \ cdots > a 1,n \\
a 2,1 > a 2,2 > \ cdots > a 2,n \\
\ vdots > \ vdots > \ ddots > \ vdots \\
a_{m,1} > a_{m,2} > \ cdots > a_{ m,n }
   \ end { pmatrix }$
```

Describing the full markup syntax is beyond the scope of this tutorial. In the next chapter, we will see how to save your work.

3.Google Colab : saving your work

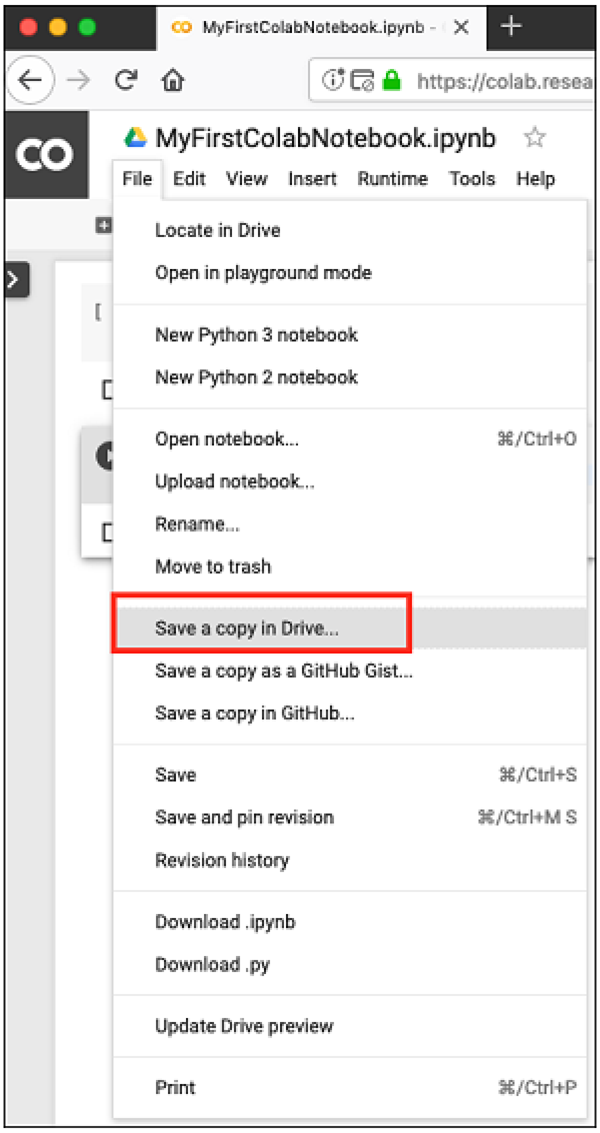
Colab allows you to save your work to Google Drive or even directly to your GitHub repository.

3.1. Saving to Google Drive

Colab allows you to save your work to your Google Drive. To save your notebook, select the following menu options:

```
File / Save a copy in Drive…
```

You will see the following screen :



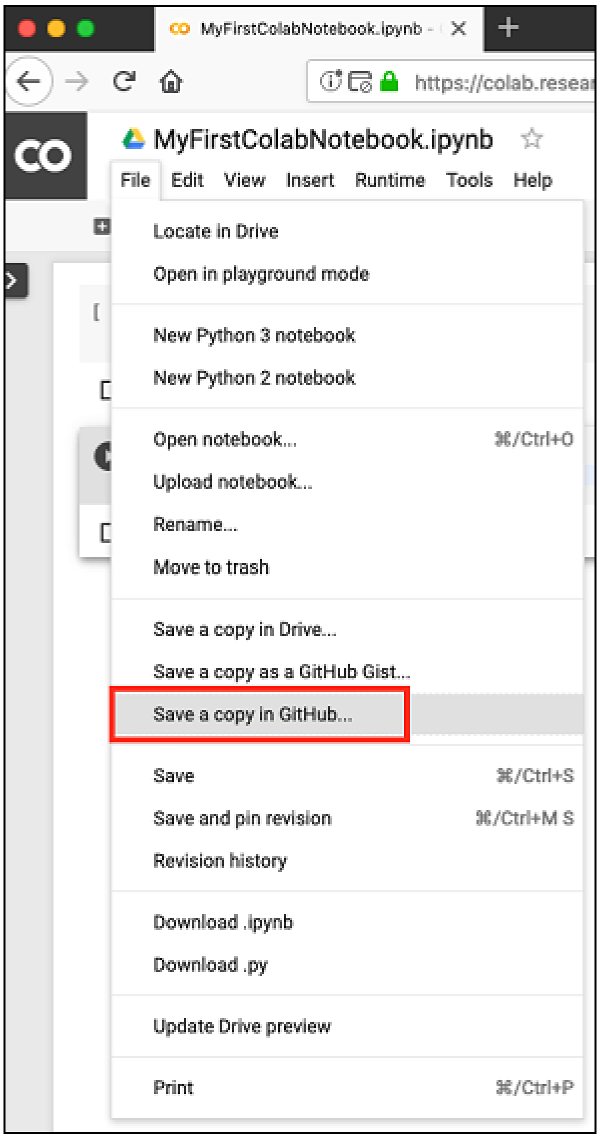
The action will create a copy of your laptop and save it to your drive. Later on, you can rename the copy to your choice of name.

3.2. Saving to GitHub

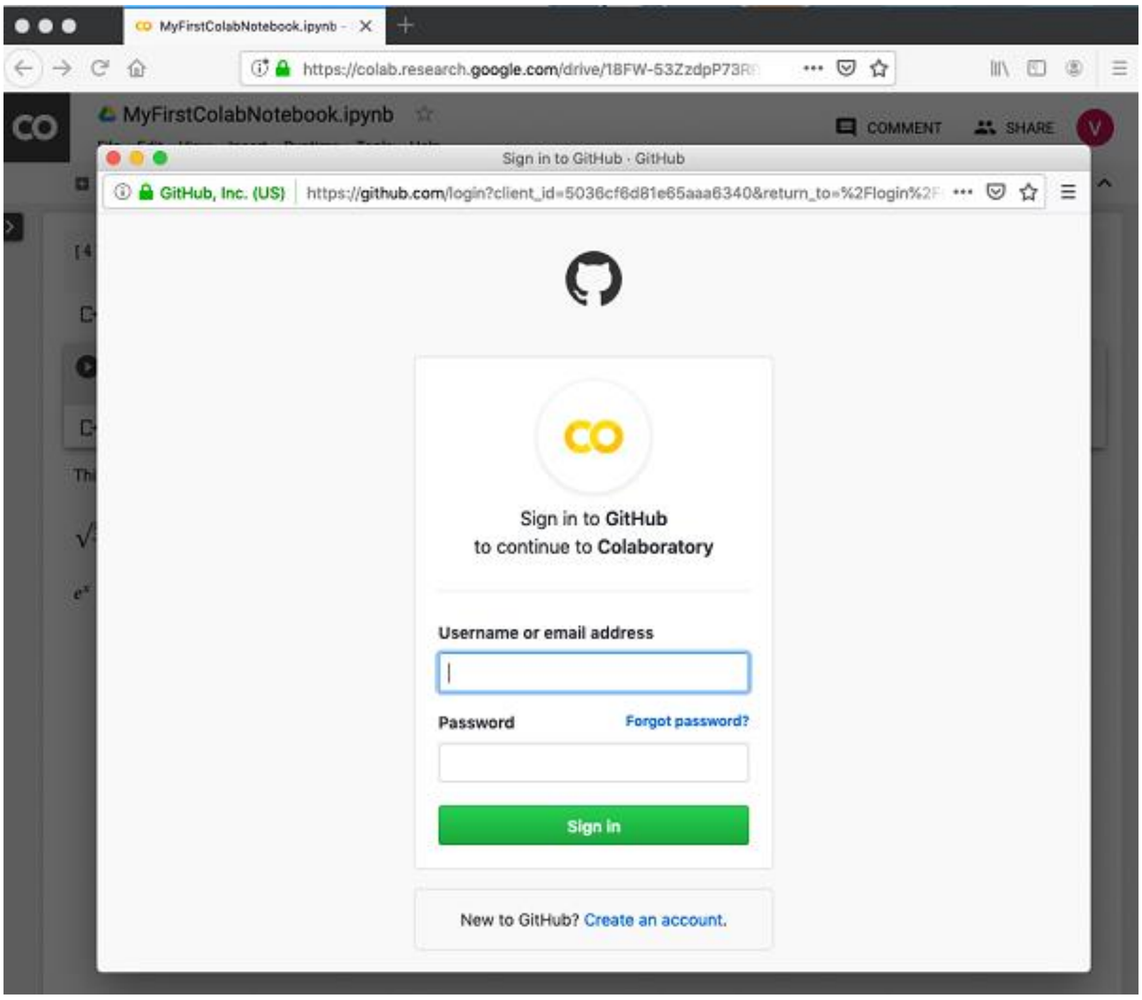
You can also save your work to your GitHub repository by selecting the following menu options:

```
File / Save a copy in GitHub...
```

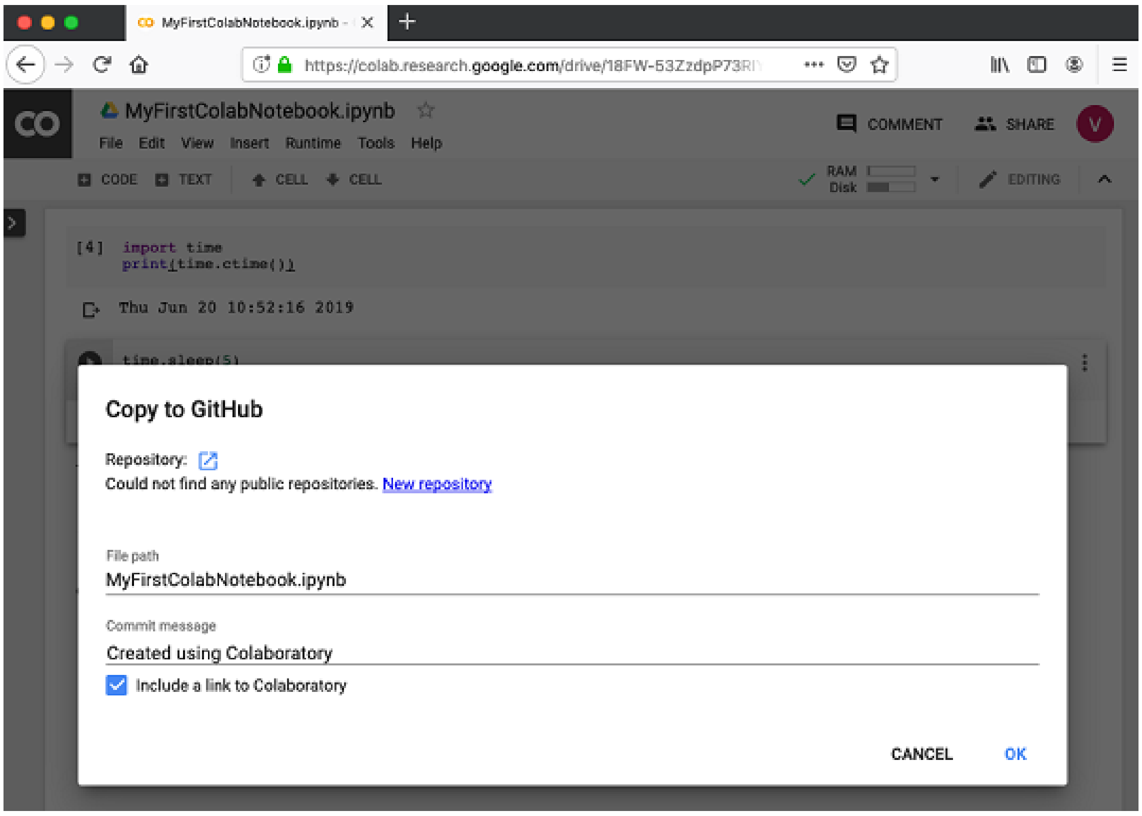
The menu selection is shown in the following screenshot for your quick reference:



You'll have to wait until you see the GitHub login screen.


MyFirstColabNotebook.ipynb
https://colab.research.google.com/drive/18FW-53ZzdpP73R
COMMENT
SHARE
Sign in to GitHub · GitHub
GitHub, Inc. (US)
https://github.com/login?client_id=5036cf6d81e65aaa6340&return_to=%2Flogin%2F
Sign in to GitHub
to continue to Colaboratory
Username or email address
Password
Forgot password?
Sign in
New to GitHub? Create an account.

Now enter your credentials. If you don't have a repository, create a new one and save your project as shown in the screenshot below:


MyFirstColabNotebook.ipynb

https://colab.research.google.com/drive/18FW-53ZzdpP73RI

```
[4] import time
    print(time.ctime())

Thu Jun 20 10:52:16 2019

time.sleep(5)
```

Copy to GitHub

Repository:
Could not find any public repositories. New repository

File path
MyFirstColabNotebook.ipynb

Commit message
Created using Colaboratory

[✓] Include a link to Colaboratory

In the next chapter, we will learn how to share your work with others.

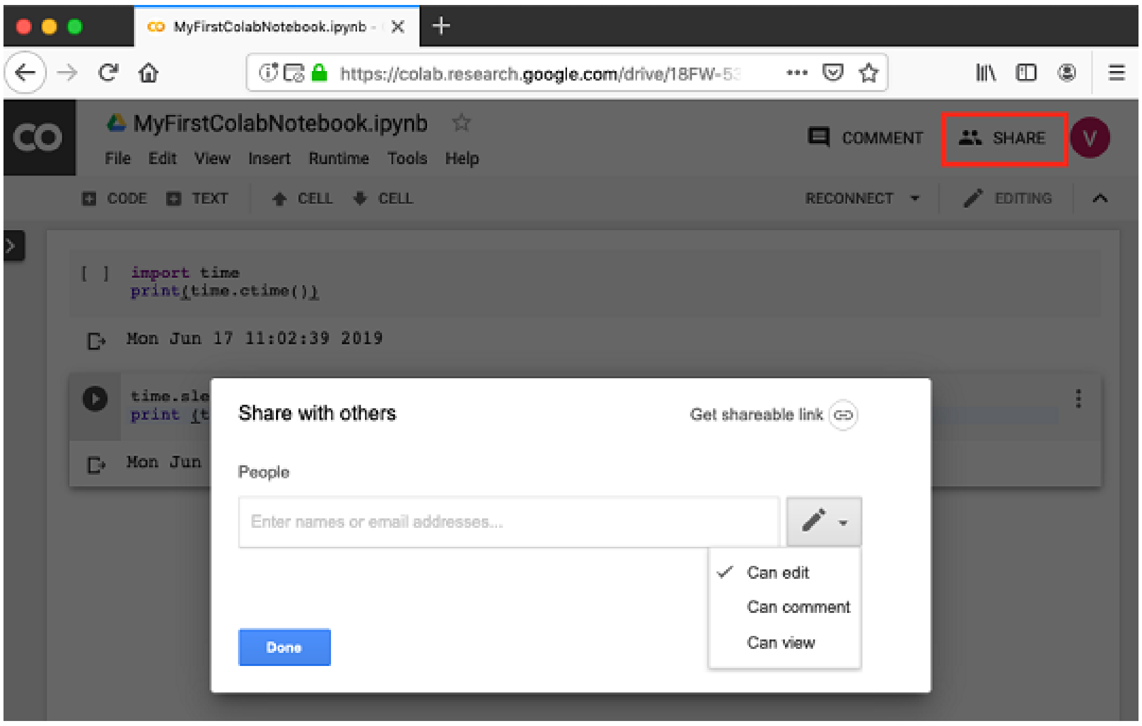
Universitat de Barcelona

4.Google Colab : share notebook

To share the notebook you've created with other co - developers, you can share the copy you've made to your Google Drive.

To publish the notebook to the general public, you can share it from your GitHub repository.

There is one more way to share your work and that is by clicking the **SHARE** link in the top right corner of your Colab notebook . This will open the sharing box as shown here:



MyFirstColabNotebook.ipynb
https://colab.research.google.com/drive/18FW-5
CO
MyFirstColabNotebook.ipynb
File Edit View Insert Runtime Tools Help
COMMENT
SHARE
V
CODE
TEXT
CELL
CELL
RECONNECT
EDITING
Share with others
Get shareable link
People
Enter names or email addresses...
Can edit
Can comment
Can view
Done

```
[ ] import time
    print(time.ctime())

    Mon Jun 17 11:02:39 2019

    time.sle
    print (t

    Mon Jun
```

You can enter the email IDs of the people you would like to share the current document with. You can set the type of access by selecting from the three options shown on the screen above.

Click on the **Get shareable link** option to get the URL of your laptop. You will find options for who to share as follows:

- Cluster people specific
- companions of his organization
- anyone with the link
- All public on the web

Now. knows how to create/run/save/share a notebook. In the Code cell, we used Python so far. The code cell can also be used to invoke system commands. This is explained below.

5.Google Colab : invoking system commands

Jupyter includes shortcuts for many common system operations. The Colab cell Code supports this feature.

5.1. simple commands

Enter the following code in the Code cell used by the echo system command.

```
message = 'A Great Tutorial on Colab by Tutorialspoint !'
greeting = !echo -e '$message\ n$message '
greeting
```

Now if you run the cell, you will see the following output:

```
['A Great Tutorial on Colab by Tutorialspoint !', 'A Great Tutorial on
Colab by Tutorialspoint !']
```

5.2. Cloning the Git repository

You can clone the entire GitHub repository to Colab using the **git** I send. For example, to clone the keras tutorial , type the following command in the Code cell:

```
!git clone https://github.com/wxs/keras-mnist-tutorial.git
```

After a successful execution of the command, you will see the following output:

```
Cloning into ' keras - mnist - tutorial'...
remote: Enumerating objects: 26, done.
remote: Total 26 (delta 0), reused 0 (delta 0), pack-reused 26
unpacking objects : 100% (26/26), done.
```

Once the repository is cloned, locate a Jupyter project (for example, MINST in keras.ipyab) in it, right-click the file name, and select the **Open With / Collaboratory** menu option to open the project in Colab .

5.3. system aliases

To get a list of shortcuts for common operations, run the following command:

```
! ls / bin
```

You will see the list in the output window as shown below:

```
bash* journalctl * sync*
bunzip2* kill* systemctl *
bzcat * kmod *
less* systemd -ask-password*
bzdiff * lessecho * systemd -escape*
systemd-hwdb *
```

Run any of these commands just like we did for **echo** and **wget** . In the next chapter, we will see how to run your previously created Python code.

Universitat de Barcelona

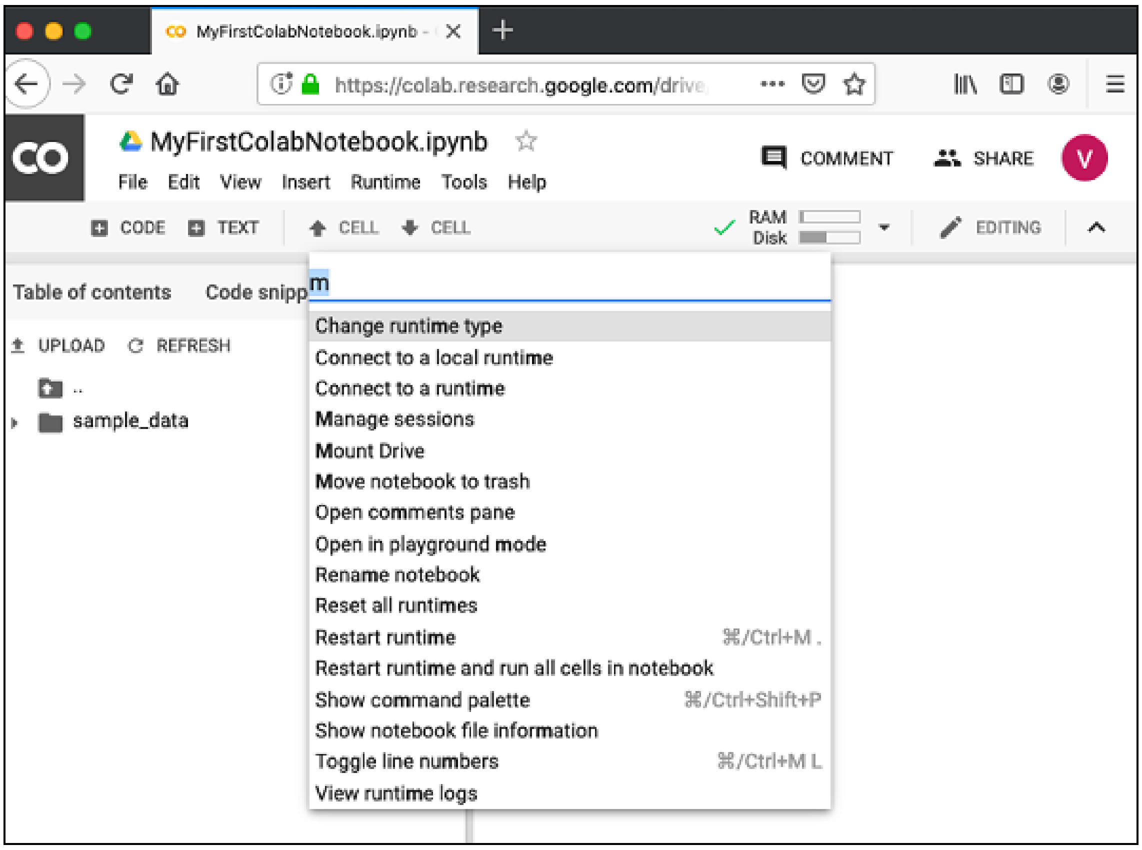
6.Google Colab - Running Python External Files

Suppose you already have developed Python code that is stored in your Google Drive. Now, you'd like to upload this code to Colab for further modification. In this chapter we will see how to load and execute the code stored in your Google Drive.

6.1. mounting unit

```
Tools / Command palette
```

You will see the list of commands as shown in this screenshot:


MyFirstColabNotebook.ipynb
https://colab.research.google.com/drive
File Edit View Insert Runtime Tools Help
COMMENT
SHARE
CODE
TEXT
CELL
CELL
RAM
Disk
EDITING
Table of contents
Code snipp
UPLOAD
REFRESH
sample_data

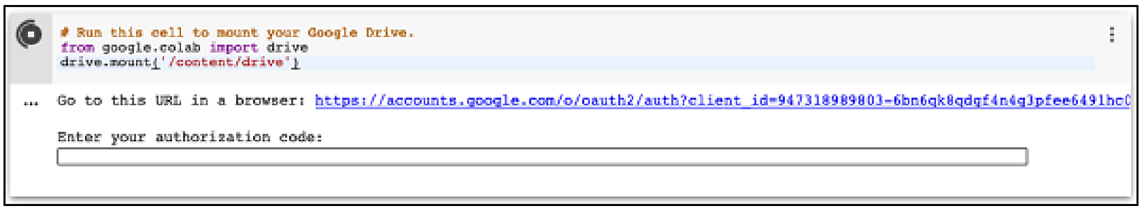
m

Change runtime type
Connect to a local runtime
Connect to a runtime
Manage sessions
Mount Drive
Move notebook to trash
Open comments pane
Open in playground mode
Rename notebook
Reset all runtimes
Restart runtime ⌘/Ctrl+M .
Restart runtime and run all cells in notebook
Show command palette ⌘/Ctrl+Shift+P
Show notebook file information
Toggle line numbers ⌘/Ctrl+M L
View runtime logs

Type some letters like "m" in the search box to locate the mount command. Select **Mount Drive** command from the list. The following code would be inserted into your Code cell.

```
# Run this cell to mount your Google Drive.
from google.colab import drive
drive.mount ('/content/drive')
```

If you run this code, you will be prompted to enter the authentication code. The corresponding screen looks as shown below:

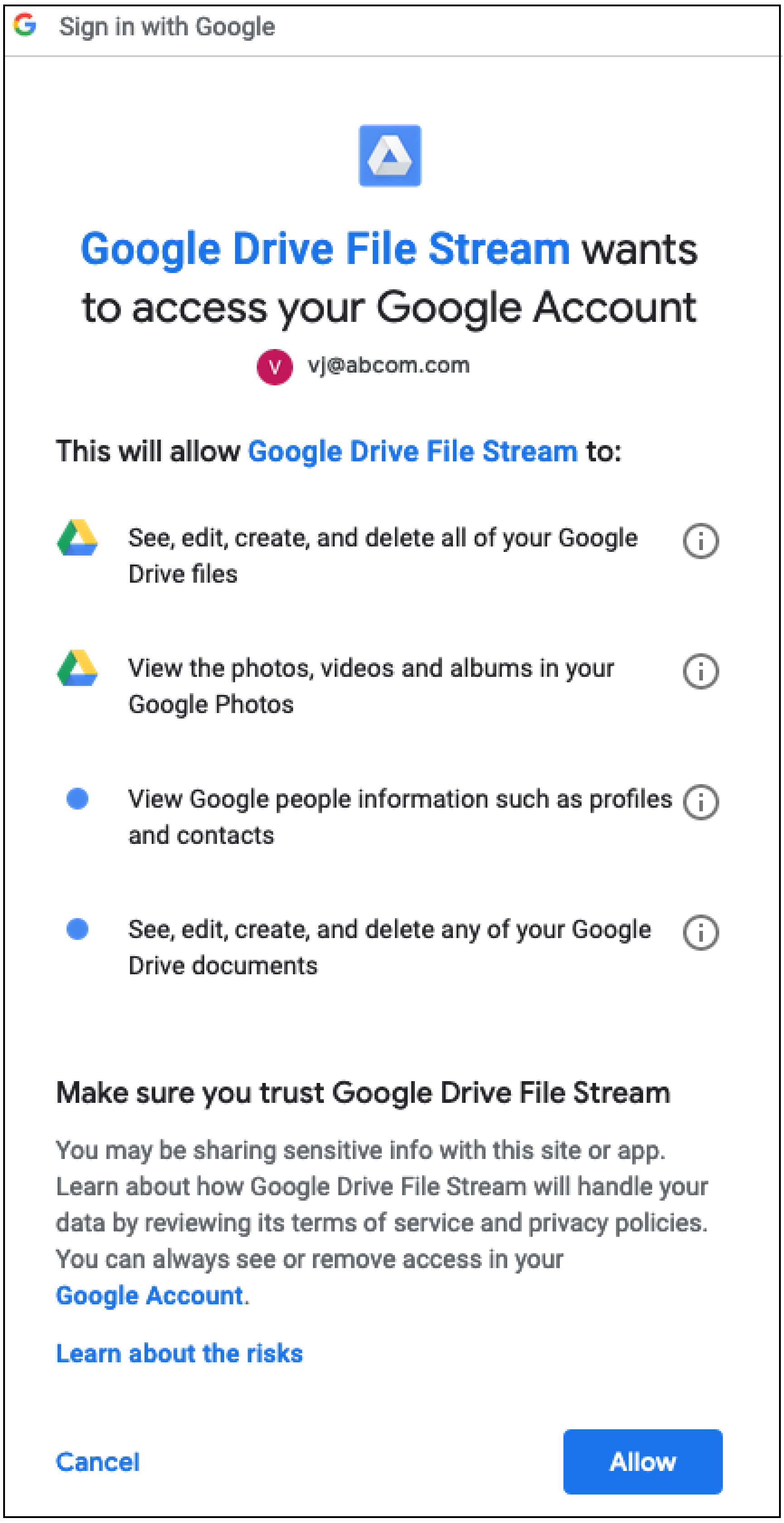


```
# Run this cell to mount your Google Drive.
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0

Enter your authorization code:
```

Open the above URL in your browser. You will be prompted to sign in to your Google account. Now, you will see the following screen:



Sign in with Google

Google Drive File Stream wants to access your Google Account

vj@abcom.com

This will allow Google Drive File Stream to:

- See, edit, create, and delete all of your Google Drive files
- View the photos, videos and albums in your Google Photos
- View Google people information such as profiles and contacts
- See, edit, create, and delete any of your Google Drive documents

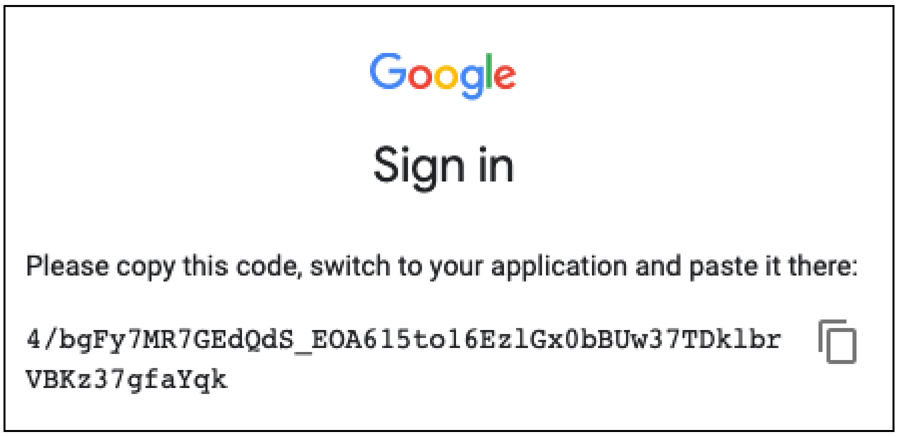
Make sure you trust Google Drive File Stream

You may be sharing sensitive info with this site or app. Learn about how Google Drive File Stream will handle your data by reviewing its terms of service and privacy policies. You can always see or remove access in your Google Account.

Learn about the risks

Cancel Allow

If you grant the permissions, you will receive your code as follows:



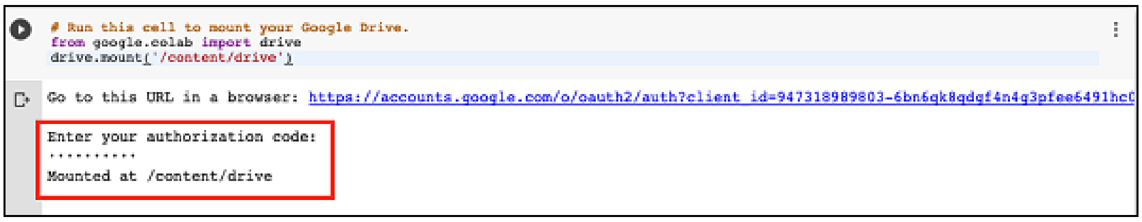
Google

Sign in

Please copy this code, switch to your application and paste it there:

```
4/bgFy7MR7GEdQdS_EOA615to16EzlGx0bBUw37TDklbr
VBKz37gfaYqk
```

Cut and paste this code into the Code cell and press ENTER. After some time, the drive will mount as seen in the screenshot below:



```
# Run this cell to mount your Google Drive.
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0

Enter your authorization code:
··········
Mounted at /content/drive
```

Now, you're ready to use the contents of your drive in Colab .

6.2. List of unit contents

You can list the contents of the drive using the ls command as follows:

```
!ls "/content/drive/My Drive/ Colab Notebooks"
```

This command will list the contents of your Colab Notebooks folder. The sample output of my drive content is shown here:

```
Greeting.ipynb hello.py LogisticRegressionCensusData.ipynb
LogisticRegressionDigitalOcean.ipynb MyFirstColabNotebook.ipynb
SamplePlot.ipynb
```

6.3. Running Python code

Now let's say you want to run a Python file called hello.py stored on your Google Drive. Type the following command in the Code cell:

```
!python 3 "/content/drive/My Drive/ Colab Notebooks/hello.py"
```

The content of hello.py is provided here for your reference:

```
print ( " Welcome to TutorialsPoint !")
```

You will now see the following output:

```
Welcome to TutorialsPoint !
```

In addition to text output, Colab also supports graphical output. We will see this in the next chapter.

7.Google Colab : graphical outputs

Colab also supports rich output as graphs. Type the following code in the Code cell.

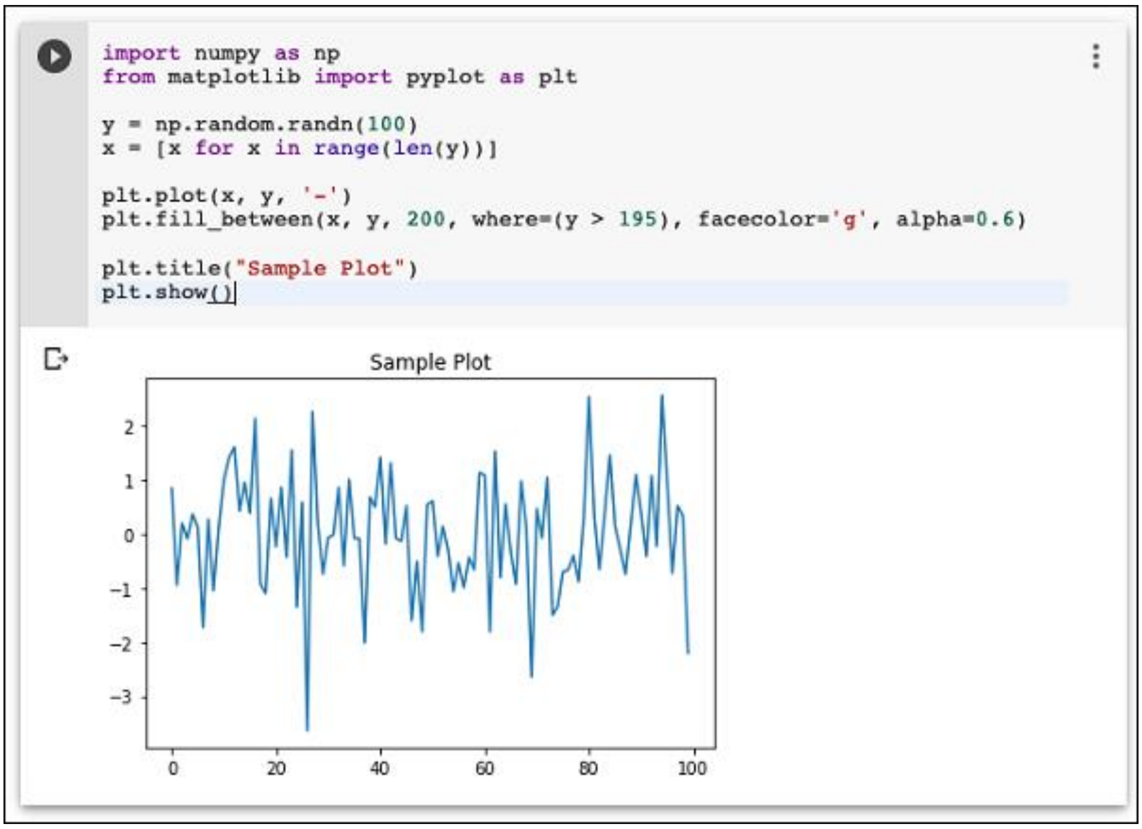
```
import numpy as np
from matplotlib import pyplot as plt

y = np.random.randn ( 100)
x = [x for x in range( len (y))]

plt.plot (x, y, '-')
plt.fill _between (x, y, 200, where = (y > 195), facecolor ='g',
alpha=0.6)

plt.title ("Sample Plot")
plt.show ()
```

Now if you run the code, you will see the following output:



```
import numpy as np
from matplotlib import pyplot as plt

y = np.random.randn(100)
x = [x for x in range(len(y))]

plt.plot(x, y, '-')
plt.fill_between(x, y, 200, where=(y > 195), facecolor='g', alpha=0.6)

plt.title("Sample Plot")
plt.show()
```

Note that the graphical output is displayed in the output section of the Code cell. Also, you will be able to create and display various types of graphics throughout your program code.

Now that you're familiar with the basics of Colab , let's move on to the Colab features that make it easy to develop your Python code.

8.Google Colab : code editing help

Today's developers rely heavily on context-sensitive help for library language and syntaxes. This is why IDEs are widely used. Colab 's notebook editor provides this functionality.

In this chapter, let's see how to request context-sensitive help when writing Python code in Colab . Follow the steps given to you whenever necessary.

8.1. Feature List

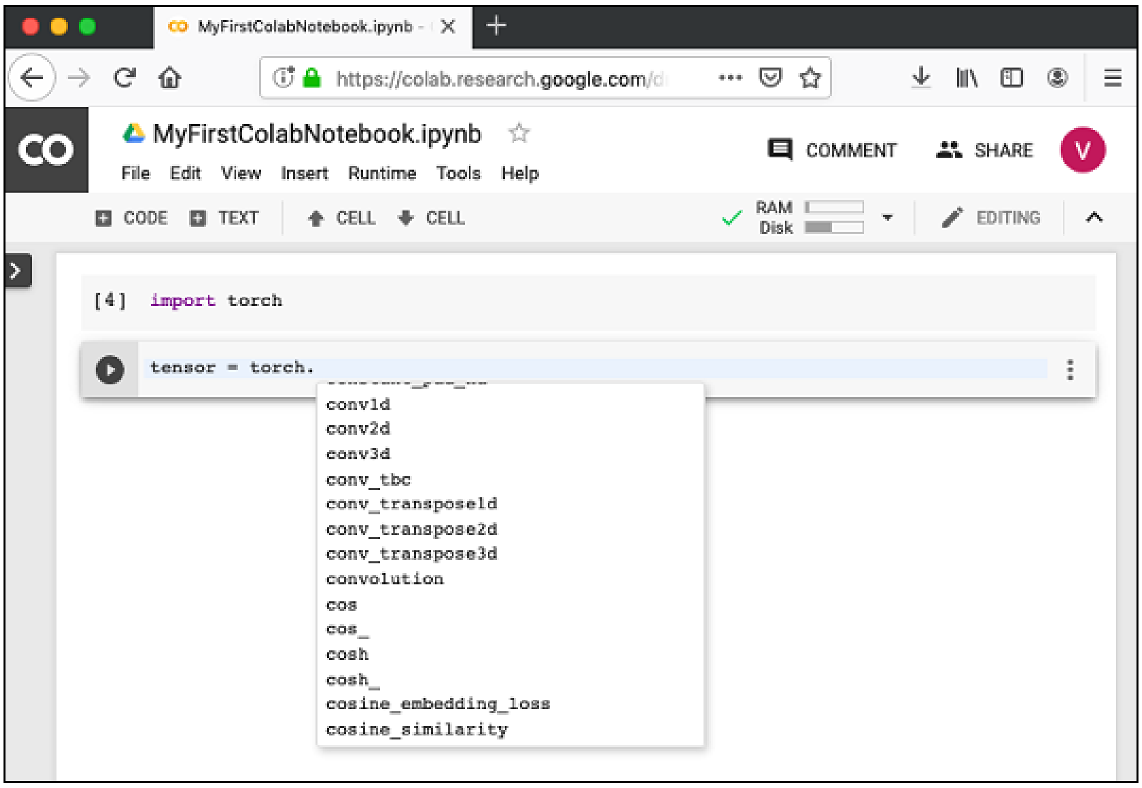
Step 1 - Open a new notebook and type the following code in the Code cell -

```
matter torch
```

Step 2 - Run the code by clicking the Run icon in the left pane of the Code cell. Add another Code cell and type the following code:

```
Tensor = torch .
```

At this point, assume that you have forgotten what the various functions available in **torch are.** module. You can request context-sensitive help on function names by pressing the **TAB** key. Note the presence of the **DOT** after the **torch** keyword. Without this DOT, you will not see hover help. Your screen would look like the screenshot shown here:


MyFirstColabNotebook.ipynb
https://colab.research.google.com/d
File Edit View Insert Runtime Tools Help
COMMENT
SHARE
CODE TEXT CELL CELL
RAM Disk
EDITING

```
[4] import torch

tensor = torch.
```

```
conv1d
conv2d
conv3d
conv_tbc
conv_transpose1d
conv_transpose2d
conv_transpose3d
convolution
cos
cos_
cosh
cosh_
cosine_embedding_loss
cosine_similarity
```

Now, select the desired feature from the list and proceed with your coding.

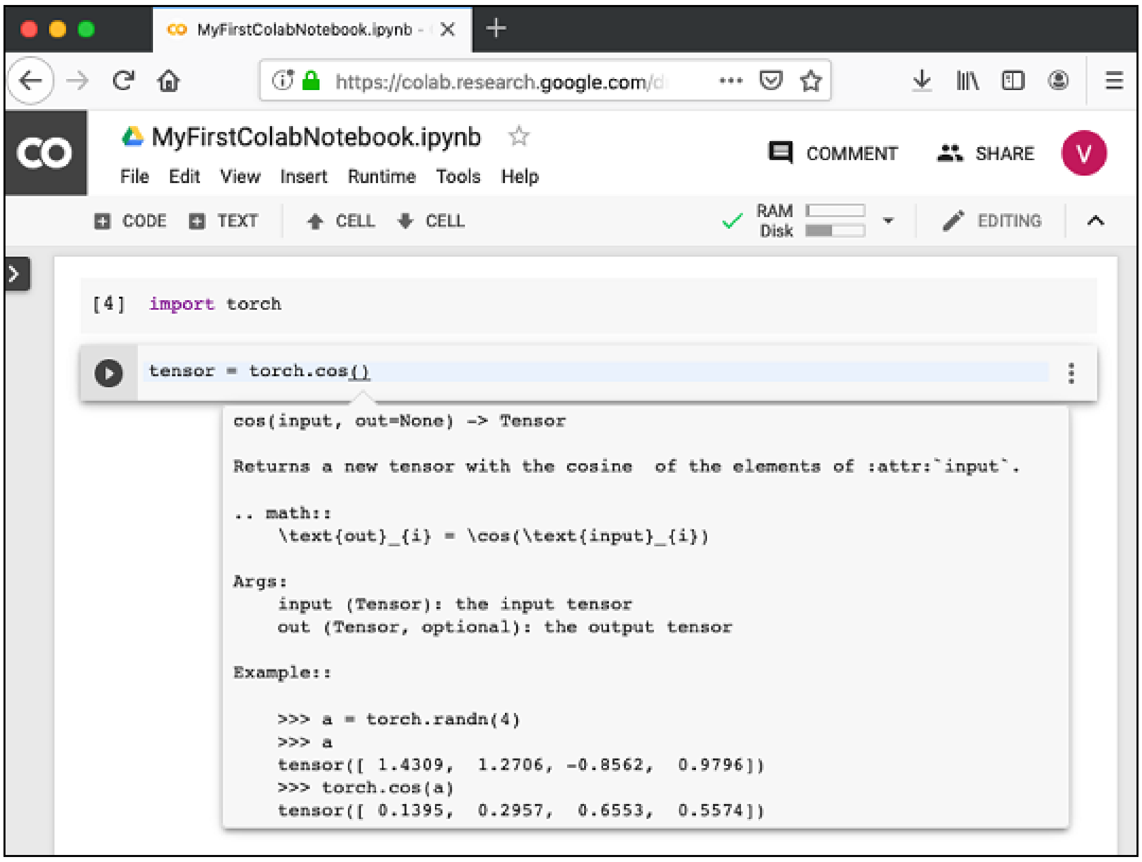
8.2. Feature Documentation

Colab gives you the documentation about any **function** or **class** as context-sensitive help.

Write the following code in your code window:

```
Tensor = torch.cos (
```

Now, hit **TAB** and you will see the documentation on **cos** in the popup window as shown on the screenshot here. Note that you must type **open parenthesis** before pressing TAB.


MyFirstColabNotebook.ipynb
https://colab.research.google.com/d
MyFirstColabNotebook.ipynb
File Edit View Insert Runtime Tools Help
COMMENT
SHARE
CODE
TEXT
CELL
CELL
RAM
Disk
EDITING

```
[4] import torch

tensor = torch.cos()
```

```
cos(input, out=None) -> Tensor

Returns a new tensor with the cosine  of the elements of :attr:`input`.

.. math::
    \text{out}_{i} = \cos(\text{input}_{i})

Args:
    input (Tensor): the input tensor
    out (Tensor, optional): the output tensor

Example::

    >>> a = torch.randn(4)
    >>> a
    tensor([ 1.4309,  1.2706, -0.8562,  0.9796])
    >>> torch.cos(a)
    tensor([ 0.1395,  0.2957,  0.6553,  0.5574])
```

In the next chapter, we'll look at **Magics** in Colab that allow us to do more powerful things than we did with system aliases.

9.Google Colab : Magic

Magics is a set of system commands that provide an extensive mini command language.

Magics are of two types:

- line magic _
- Magic cell phone

Line magic as its name suggests consists of a single command line, while cell magic covers the entire cell body of code.

For line magic, the command is prepended with a single % character and for cell magic, it is prepended with two % characters (%%).

Let's look at some examples of both to illustrate them.

9.1. line magic

Write the following code in your code cell:

```
% ldir
```

You will see the contents of your local directory, something like this:

```
drwxr - xr -x 3 root 4096 Jun 20 10:05 drive/
drwxr -xr -x 1 root 4096 May 31 16:17 sample_data /
```

Try the following command:

```
% history
```

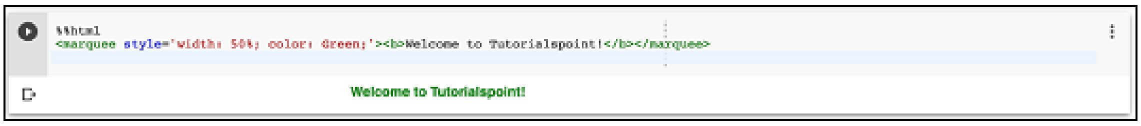
This presents the full history of the commands you have previously executed.

9.2. cell magic

Write the following code in your code cell:

```
%%html
<marquee style='width: 50%; color: Green;'>Welcome to Tutorialspoint
!< /marquee>
```

Now if you run the code and you will see the welcome message scrolling on the screen as shown here:



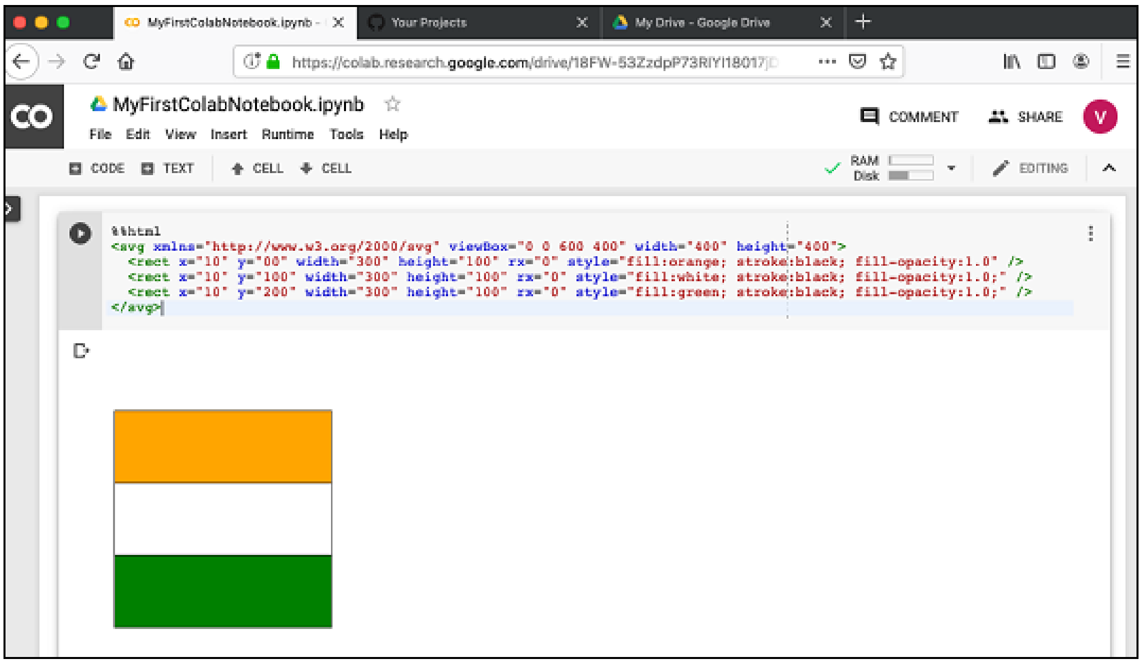
```
%%html
<marquee style='width: 50%; color: green;'><b>Welcome to Tutorialspoint!</b></marquee>
```

```
Welcome to Tutorialspoint!
```

The following code will add SVG to your document.

```
%%html
< svg xmlns = "https://www.w3.org/2000/svg" viewBox ="0 0 600 400"
width="400" height="400">
< rect x="10" y="00" width="300" height="100" rx ="0" style="
fill:orange ; stroke:black ; fill-opacity:1.0" />
< rect x="10" y="100" width="300" height="100" rx ="0" style="
fill:white ; stroke:black ; fill-opacity:1.0;" />
< rect x="10" y="200" width="300" height="100" rx ="0" style="
fill:green ; stroke:black ; fill-opacity:1.0;" />
</ svg >
```

If you run the code, you will see the following output:


MyFirstColabNotebook.ipynb
Your Projects
My Drive - Google Drive
https://colab.research.google.com/drive/18FW-53ZzdpP73RIYI18017j
MyFirstColabNotebook.ipynb
File Edit View Insert Runtime Tools Help
COMMENT
SHARE
CODE
TEXT
CELL
CELL
RAM
Disk
EDITING

```
%%html
<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 600 400" width="400" height="400">
  <rect x="10" y="00" width="300" height="100" rx="0" style="fill:orange; stroke:black; fill-opacity:1.0" />
  <rect x="10" y="100" width="300" height="100" rx="0" style="fill:white; stroke:black; fill-opacity:1.0;" />
  <rect x="10" y="200" width="300" height="100" rx="0" style="fill:green; stroke:black; fill-opacity:1.0;" />
</svg>
```

9.3. Spell List

To get a full list of supported magics, run the following command:

```
% lsmagic
```

You will see the following output:

```
Available line magics :
%alias % alias_magic % autocall % automagic % autosave %bookmark %cat
%cd %clear
```

```
%colors % config % connect_info % cp %debug % dhist % dirs %
doctest_mode % ed %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
%profile % 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 %shell %store % sx %system % tb % tensorflow_version
%time % timeit
% unalias % unload_ext %who % who_ls % whos % xdel % xmode

Available cell magics :
%%! %%HTML %%SVG %%bash %% bigquery %%capture %%debug %%file %%html %%
javascript
%% js %%latex %% perl %% prun %% pypy %%python %%python2 %%python3
%%ruby %%script
%% sh %%shell %% svg %% sx %%system %%time %% timeit %% writefile

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

Next, you will learn another powerful feature in Colab to set program variables at runtime.

Universitat de Barcelona

10. Google Colab : add forms

Colab provides a very useful utility called Forms that allows you to accept user input at runtime. Now let's see how to add forms to your notebook.

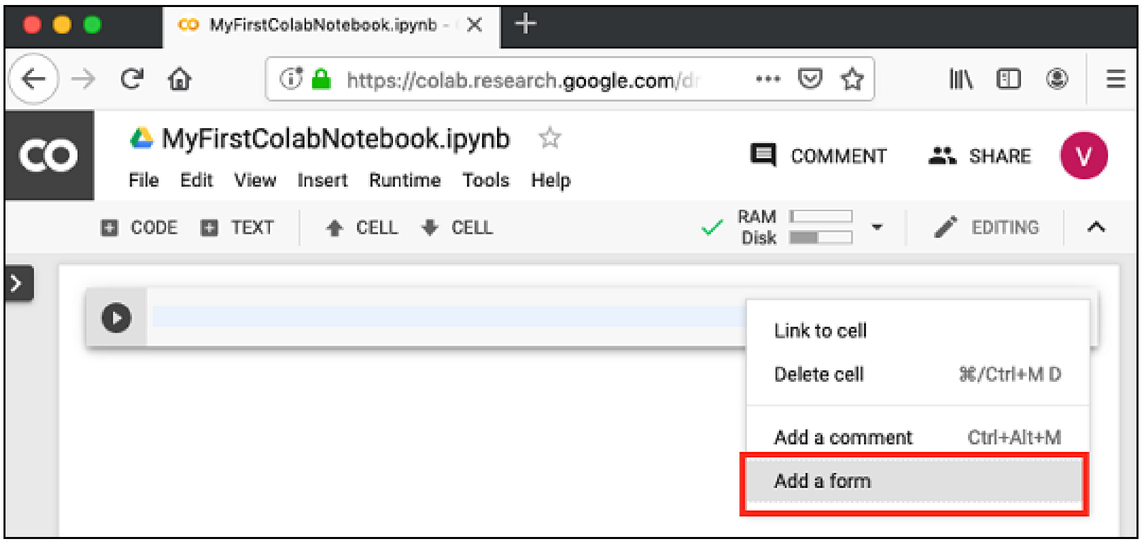
10.1. add form

In a previous lesson, you used the following code to create a time delay:

```
import time
print( time.ctime ())
time.sleep (5)
print ( time.ctime ())
```

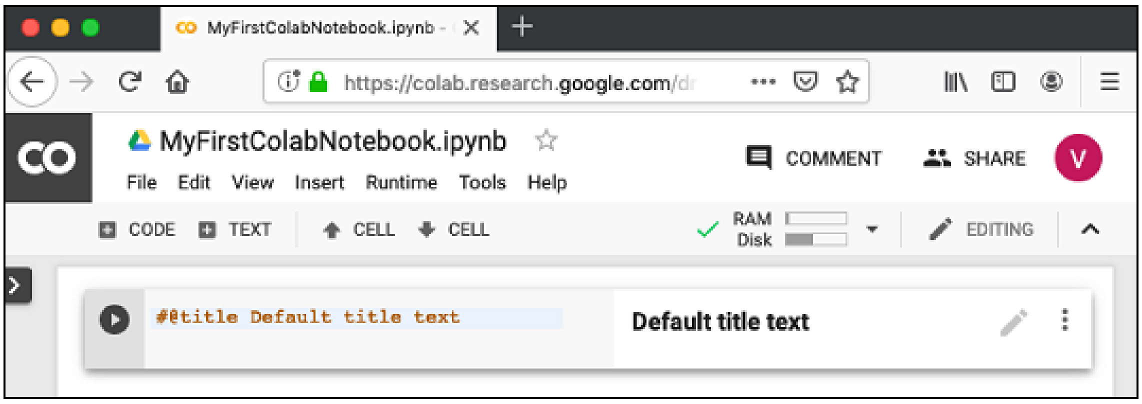
Let's say you want a user-set time delay instead of a fixed 5-second delay. To do this, you can add a form to the Code cell to accept the sleep time.

Open a new notebook. Click on the **Options** menu (with vertical dots). A pop-up menu appears as seen on the screenshot below:



Now select **Add a form** option. It will add the form to your Code cell with a default title as seen on the screenshot here:

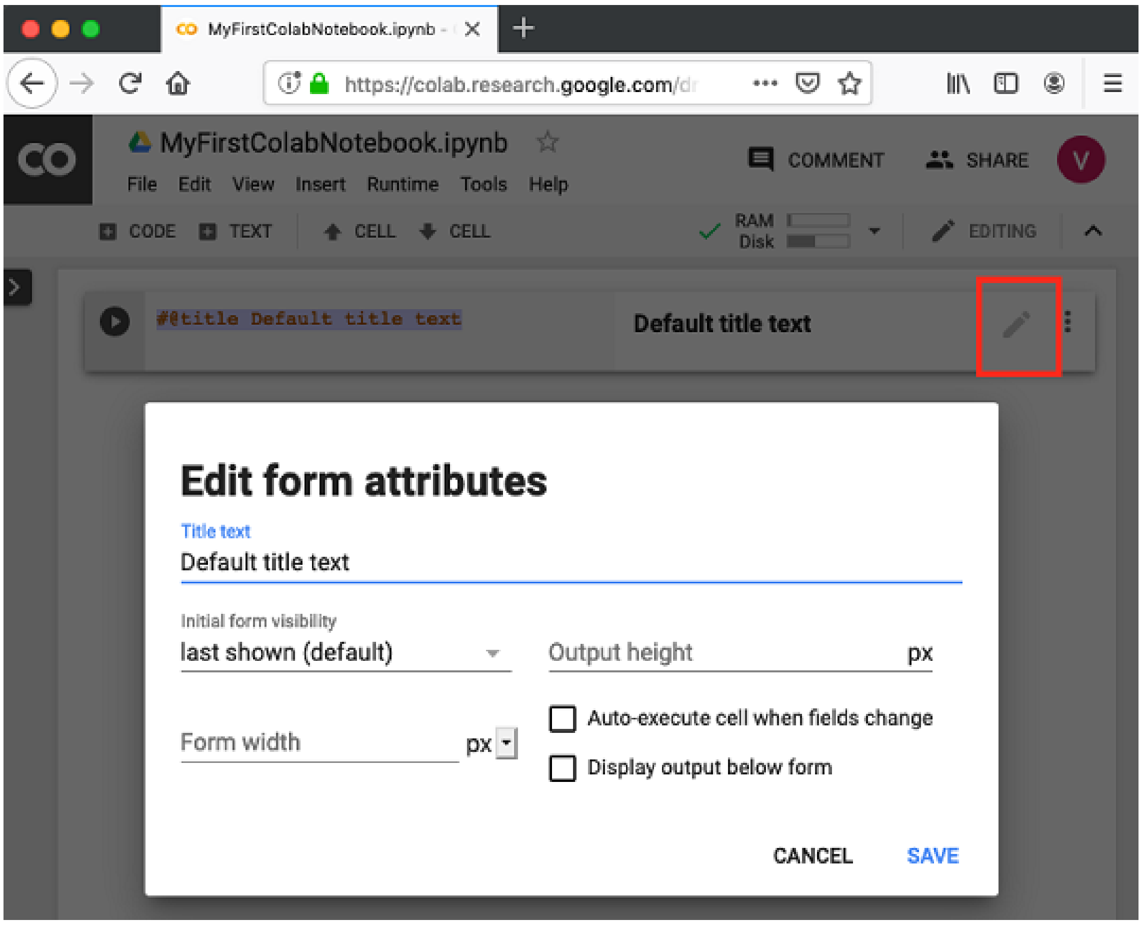
Universitat de Barcelona


MyFirstColabNotebook.ipynb
https://colab.research.google.com/d
COMMENT
SHARE
File Edit View Insert Runtime Tools Help
CODE TEXT CELL CELL
RAM Disk
EDITING

```
#@title Default title text
```

Default title text

To change the title of the form, click on the **Settings** button (pencil icon on the right). A configuration screen will appear as shown here:


MyFirstColabNotebook.ipynb
https://colab.research.google.com/d
COMMENT
SHARE
File Edit View Insert Runtime Tools Help
CODE TEXT CELL CELL
RAM Disk
EDITING

```
#@title Default title text
```

Default title text

Edit form attributes

Title text: Default title text

Initial form visibility: last shown (default)

Output height px

Form width px

☐ Auto-execute cell when fields change

☐ Display output below form

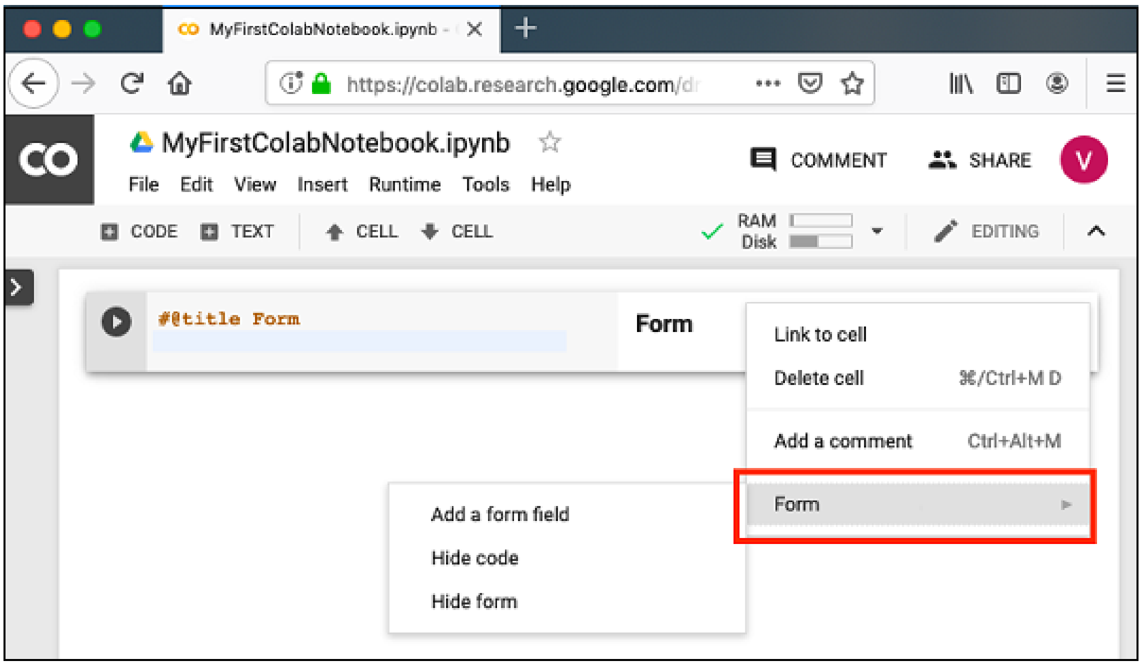
CANCEL SAVE

Change the form title to " **Form** " and save the form. You can use another name of your choice. Notice that it adds the **@title to** its code cell.

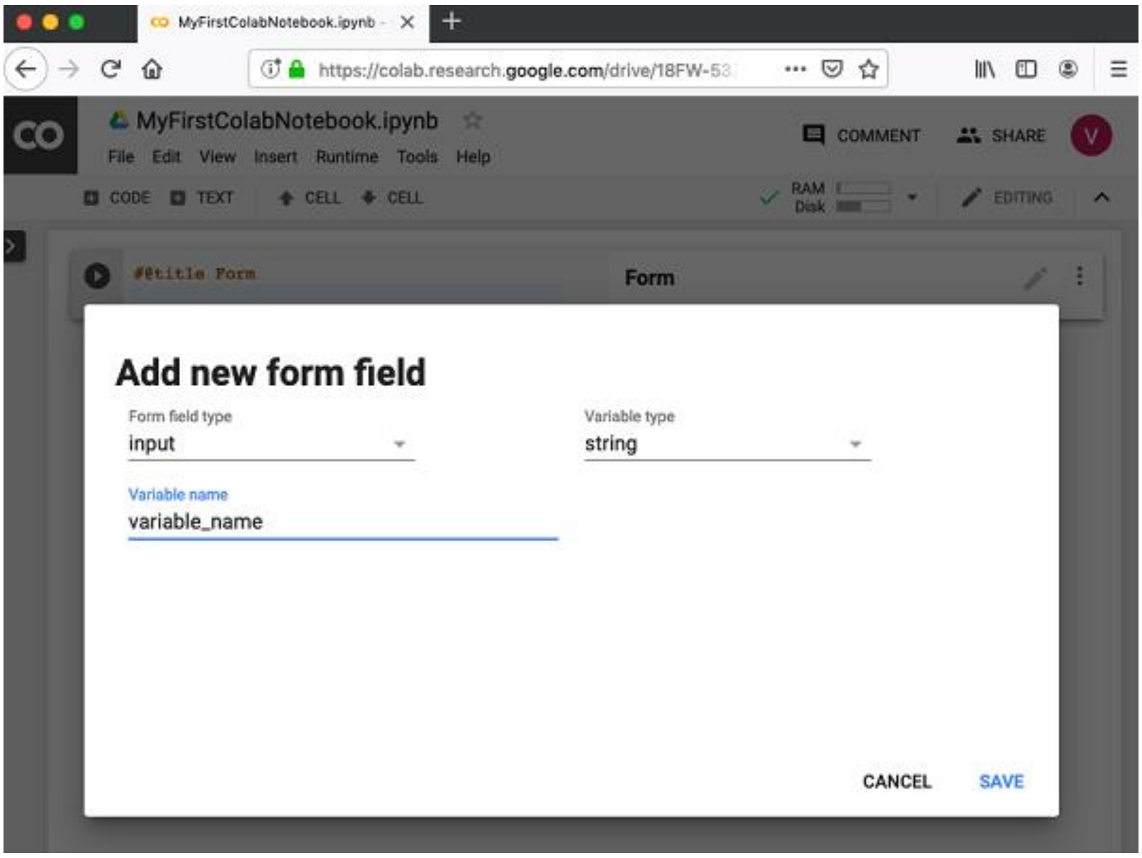
You can explore other options on the previous screen later. In the next section, we will learn how to add input fields to the form.

10.2. Add form fields

To add a form field, click the **Options** menu in the Code cell, click **Form** to reveal the submenus. The screen will look as shown below:



Select **Add a form field** menu option. A dialog appears as seen here:


MyFirstColabNotebook.ipynb
https://colab.research.google.com/drive/18FW-53
File Edit View Insert Runtime Tools Help
COMMENT SHARE
CODE TEXT CELL CELL
RAM Disk
EDITING
CANCEL SAVE

```
#@title Form
```

Form

Add new form field

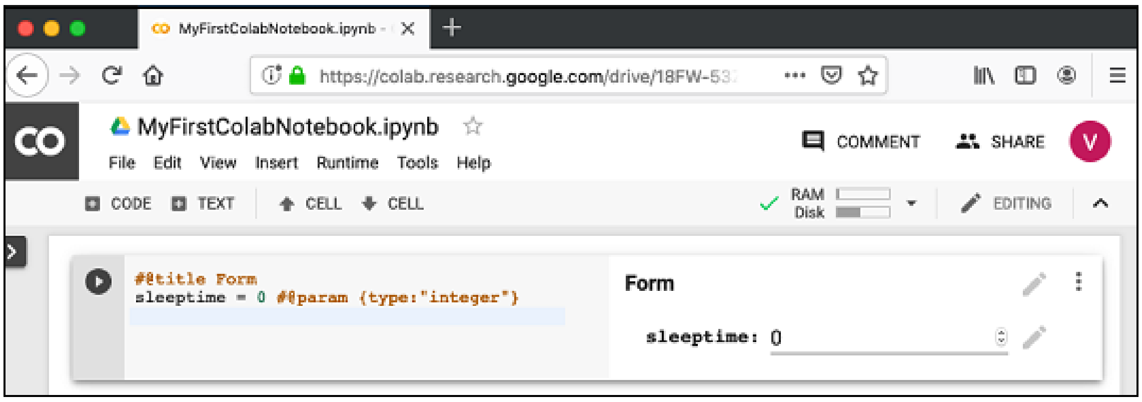
Form field type: input

Variable type: string

Variable name: variable_name

Leave the **Form field type** a **input** . Change the **Variable name** to **sleeptime** and set the **Variable type** to **integer** . Save the changes by clicking the **Save** button.

Your screen will now look like the following with the variable **sleeptime** added in the code.


MyFirstColabNotebook.ipynb
https://colab.research.google.com/drive/18FW-53
File Edit View Insert Runtime Tools Help
COMMENT SHARE
CODE TEXT CELL CELL
RAM Disk
EDITING

```
#@title Form
sleeptime = 0 #@param {type:"integer"}
```

Form

sleeptime: 0

Next, let's see how to test the form by adding code that uses the **sleeptime** variable.

10.3. test form

Add a new Code cell below the form cell. Use the code provided below:

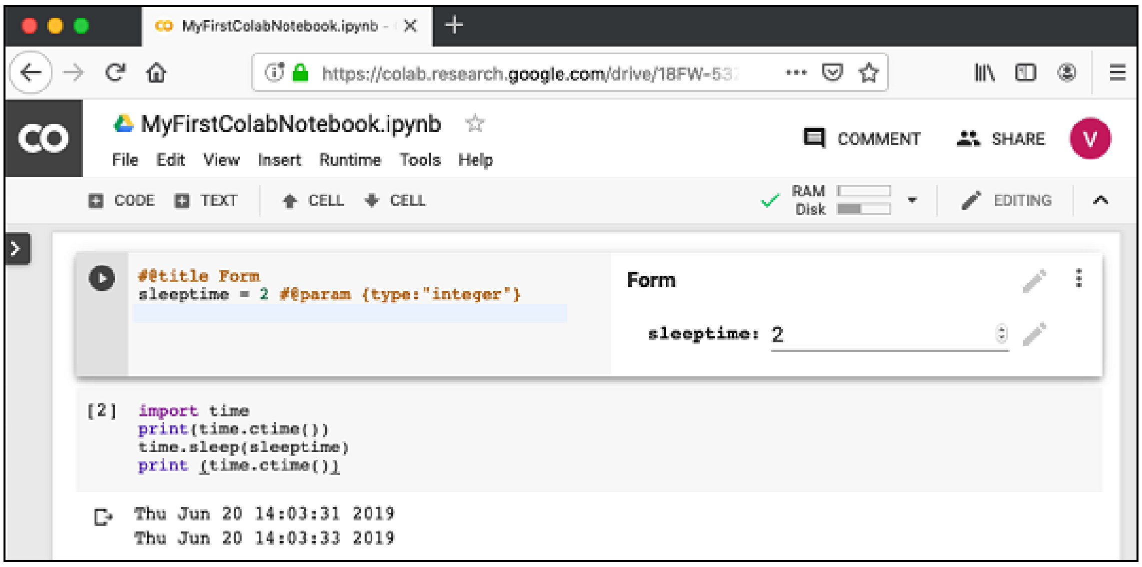
```
import time
print( time.ctime ())
time.sleep ( sleeptime )
print ( time.ctime ())
```

You used this code in the previous lesson. Prints the current time, waits a certain amount of time, and prints a new timestamp. The amount of time the program waits is set in the variable called **sleeptime** .

Now, go back to the **Form** Cell and type a value of 2 for **sleeptime** . Select the following menu -

```
Runtime / Run all
```

This runs the entire notebook. You can see an output screen as shown below.


MyFirstColabNotebook.ipynb
https://colab.research.google.com/drive/18FW-53
File Edit View Insert Runtime Tools Help
COMMENT
SHARE
CODE TEXT CELL CELL
RAM Disk
EDITING
Form

```
#@title Form
sleeptime = 2 #@param {type:"integer"}
```

sleeptime: 2

```
[2] import time
    print(time.ctime())
    time.sleep(sleeptime)
    print (time.ctime())
```

```
Thu Jun 20 14:03:31 2019
Thu Jun 20 14:03:33 2019
```

Notice that you have taken your input value of 2 for the

Bedtime

. Try changing this to a different value and **Run all** to see its effect.

10.4. Entering text

To accept text input on your form, enter the following code in a new code cell.

```
name = ' Tutorialspoint ' #@ param { type:"string "}
```

```
print ( name )
```

Now if you run the Code cell, whatever name you set on the form will be printed to the screen. By default, the following output would appear on the screen.

```
tutorialspoint
```

Note that you can use the menu options as shown for integer input to create a **Text** input field.

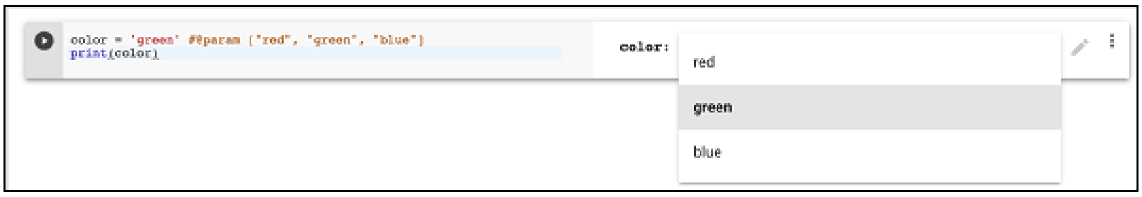
10.5. The drop down list

To add a dropdown list to your form, use the following code:

```
color = 'green' #@ param ["red", "green", "blue"]
print (color)
```

This creates a dropdown list with three values: red, green, and blue. The default selection is green.

The dropdown list is shown in the screenshot below:



```
color = 'green' #@param ["red", "green", "blue"]
print(color)
```

color:

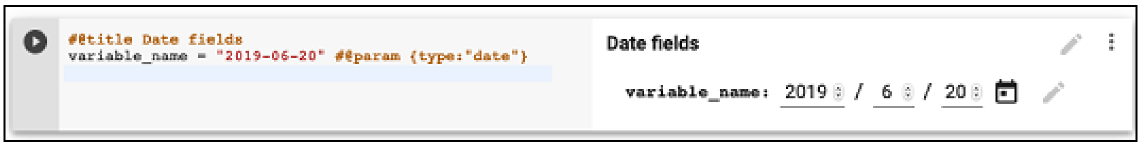
- red
- green
- blue

10.6. date entry

Colab Form allows you to accept dates in your code with validations. Use the following code to insert the date into your code.

```
#@title Date fields
date_input = '2019-06-03' #@ param { type:"date "}
print ( date_input )
```

The form screen looks like this.



```
#@title Date fields
variable_name = "2019-06-20" #@param {type:"date"}
```

Date fields

variable_name: 2019 / 6 / 20

Try entering an incorrect date value and watch the validations.

So far, you've learned how to use Colab to create and run Jupyter notebooks with your Python code. In the next chapter, we'll see how to install popular ML libraries in your notebook so you can use them in your Python code.

11. Google Colab : Installing AA Libraries

Colab is compatible with most of the machine learning libraries available on the market. In this chapter, let's take a quick overview of how to install these libraries on your Colab laptop .

To install a library, you can use any of these options:

```
!pip install
```

either

```
!apt -get install
```

11.1. Keras

Keras , written in Python, runs on top of TensorFlow , CNTK or Theano . It enables fast and easy prototyping of neural network applications. It supports both convolutional networks (CNN) and recurrent networks, and also their combinations. It is fully GPU compatible.

To install Keras , use the following command:

```
! pip install -q keras
```

11.2. PyTorch

PyTorch is ideal for developing deep learning applications. It is an optimized tensor library and is GPU enabled. To install PyTorch , use the following command:

```
!pip 3 install torch torch vision
```

11.3. MxNet

Apache MxNet is another flexible and efficient library for deep learning. To install MxNet , run the following commands:

```
!apt install libnvrtc8.0
!pip install mxnet-cu80
```

11.4. OpenCV

OpenCV is an open source computer vision library for developing machine learning applications. It has more than 2,500 optimized algorithms that support various applications, such as recognizing faces, identifying objects, tracking moving objects, stitching images, etc. Giants like Google, Yahoo , Microsoft, Intel, IBM, Sony, Honda, Toyota use this library. This is very suitable for developing real-time vision applications.

To install OpenCV use the following command:

```
!apt -get -qq install -y libsm6 libxext6 && pip install -q -U opencv -
python
```

11.5. XGBoost

XGBoost is a distributed gradient-boosting library that runs on major distributed environments like Hadoop . It is very efficient, flexible and portable. Implement ML algorithms under the Gradient framework Boost .

To install XGBoost , use the following command:

```
!pip install -q xgboost ==0.4a30
```

11.6. GraphViz

Graphviz is open source software for graph visualizations. It is used for network visualization, bioinformatics, database design, and for that matter in many domains where a visual interface to data is desired.

To install GraphViz , use the following command:

```
!apt -get -qq install -y graphviz && pip install -q pydot
```

At this point, you have learned how to create Jupyter notebooks that contain popular machine learning libraries. You are now ready to develop your machine learning models. This requires a lot of processing power. Colab offers free GPU for its laptops.

In the next chapter, we will learn how to enable GPU for your laptop.

Universitat de Barcelona

12. Google Colab : free GPU usage

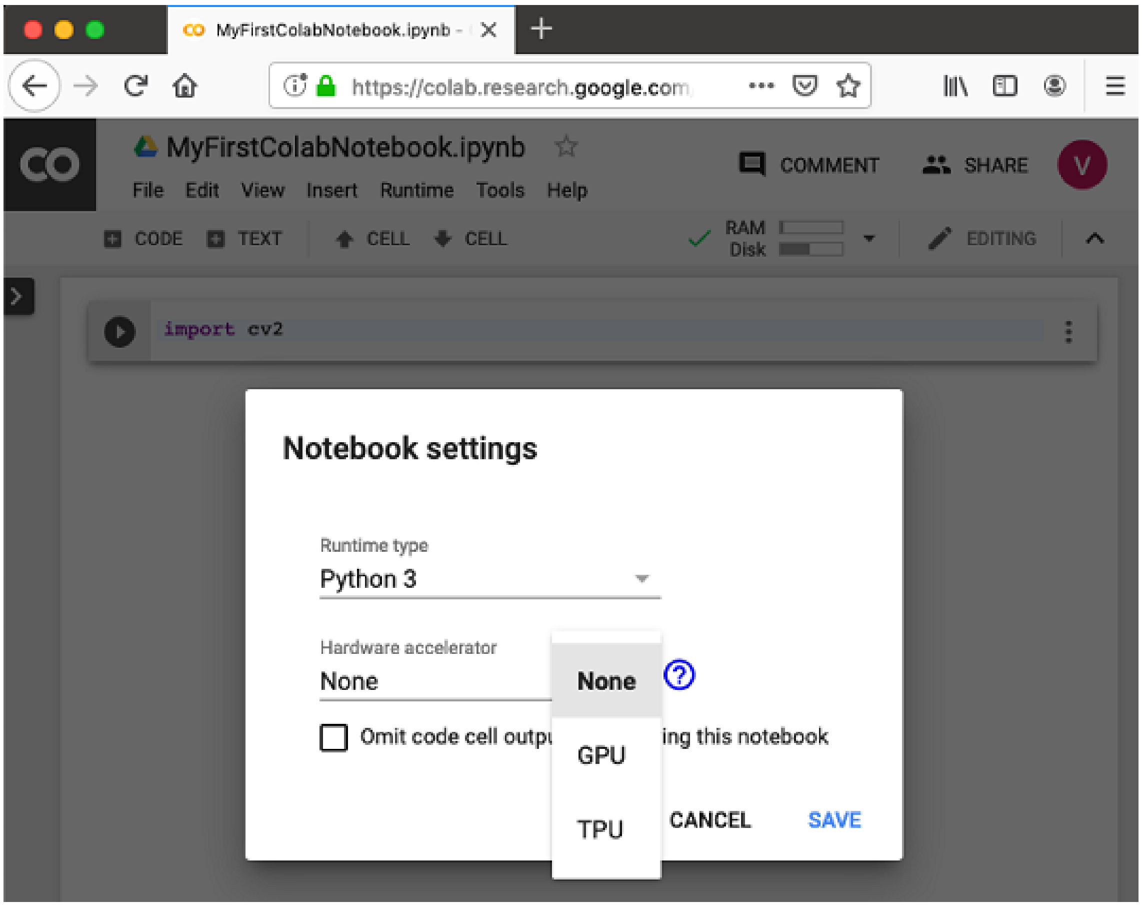
Google provides free GPU usage for its Colab laptops .

12.1. GPU enablement

To enable the GPU on your laptop, select the following menu options:

```
Runtime / Change runtime type
```

You will see the following screen What output :


MyFirstColabNotebook.ipynb
https://colab.research.google.com
COMMENT
SHARE
File Edit View Insert Runtime Tools Help
CODE TEXT CELL CELL
RAM Disk
EDITING

```
import cv2
```

Notebook settings

Runtime type: Python 3

Hardware accelerator: None

None
GPU
TPU
Omit code cell outpu... ing this notebook
CANCEL
SAVE

Select **GPU** and your laptop would use the cloud-provided free GPU during rendering. To get familiar with GPU rendering, try running the sample app from the **MNIST** tutorial that you cloned earlier.

```
!python 3 "/content/drive/MyDrive/app/mnist_cnn.py"
```

Try running the same python file without the GPU enabled. Did you notice the difference in execution speed?

12.2. GPU test

You can easily check if the GPU is enabled by running the following code:

```
matter tensorflow as tf
tf.test.gpu_device_name ( ) _
```

If the GPU is enabled, it will give the following output:

```
'/ device:GPU :0'
```

12.3. Device list

If you are curious about the devices used while running your cloud laptop, try the following code:

```
desde tensorflow.python.client import device_lib
device_lib.list_local_devices ( )
```

You will see the output of the following way :

```
[ name : "/device:CPU:0"
   device _type : "CPU"
   memory_limit : 268435456
   locality { }
   incarnation: 1734904979049303143, name : "/device:XLA_CPU:0"
   device_type : "XLA_CPU" memory_limit : 17179869184
   locality { }
   incarnation: 16069148927281628039
   physical _device_desc : " device : XLA_CPU device ", name :
"/device:XLA_GPU:0"
   device_type : "XLA_GPU"
   memory_limit : 17179869184
   locality { }
   incarnation: 16623465188569787091
   physical _device_desc : " device : XLA_GPU device ", name :
"/device:GPU:0"
   device_type : "GPU"
   memory_limit : 14062547764
locality {
      bus_id : 1
links { }
}
incarnation: 6674128802944374158
physical_device_desc : " device : 0, name : Tesla T4, pci bus id:
0000:00:04.0, compute capability : 7.5"]
```

12.4. Checking RAM

To see the memory resources available to your process, type the following command:

```
! cat / proc / meminfo
```

You will see the following output:

```
13335276 kB
MemFree : 7322964 kB
MemAvailable : 10519168 kB
Buffers: 95732 kB
Cache: 2787632KB
SwapCached : 0 kB
Active: 2433984KB
Inactive: 3060124KB
Active(anon): 2101704KB
Inactive(anon): 22880KB
Active(file): 332280KB
Inactive(file): 3037244KB
```

You are now all set for developing machine learning models in Python using Google Colab .