4C16 Lab System Handbook

Hugh Denman, Vibhoothi, François Pitié

Trinity College Dublin

2022

Foreword

When we designed this module in 2017, we knew that labs would require expensive computing resources and that software frameworks would change very quickly.

We have thus designed a unique lab system, specifically for this course, so that you can work, remotely, on a modern environment, with up to date machines, learn good industry practices and get instantaneous feedback.

Labs are written in Python, using the Keras Deep Learning framework, running on Google's Colab notebooks, which will allow you to write and execute code through the browser. Assessments will be automatically graded through our git-based web application.

Why Python?

Python offers the convenience of high-level facilities for general programming, combined with specialised high-performance libraries (which are not themselves written in Python) for number crunching.

Also:

- Free
- Reasonably friendly
- Reasonably strict
- Can be nearly industrial
- (alas?) not Matlab

Not a good language for implementing high-performance cutting-edge algorithms, but a good language for using them.

Why Python?

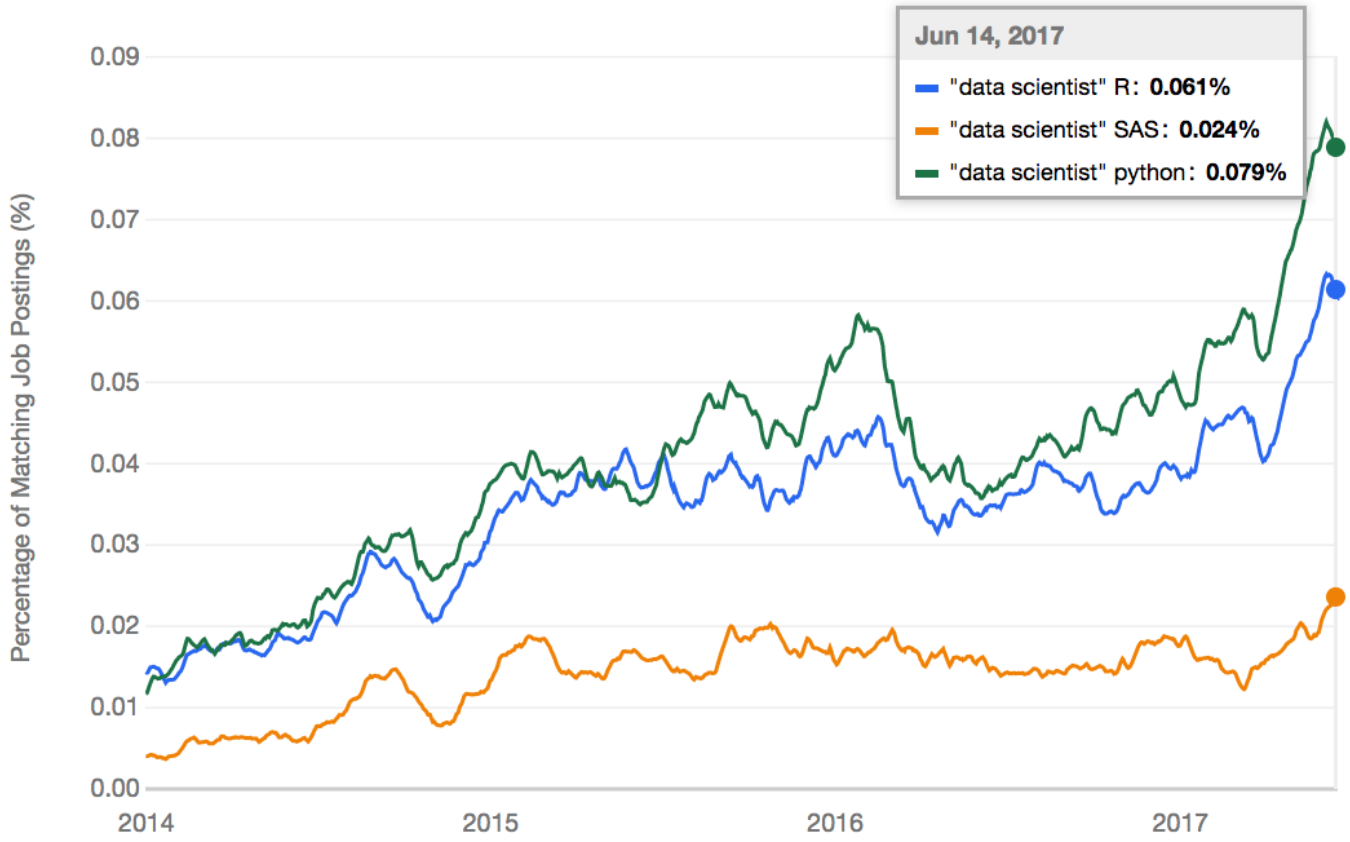
Worldwide, Sept 2022 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Python	28.29 %	-1.8 %
2		Java	17.31 %	-0.7 %
3		JavaScript	9.44 %	-0.1 %
4		C#	7.04 %	-0.1 %
5		C/C++	6.27 %	-0.4 %
6		PHP	5.34 %	-1.0 %
7		R	4.18 %	+0.3 %
8	↑↑↑	TypeScript	3.05 %	+1.5 %
9	↑↑↑	Go	2.16 %	+0.6 %
10		Swift	2.11 %	+0.5 %
11	↓↓↓	Objective-C	1.93 %	-0.0 %
12	↓↓↓	Kotlin	1.88 %	+0.0 %
13		Matlab	1.55 %	+0.1 %

source: The PYPL PopularitY of Programming Language Index is created by analyzing how often language tutorials are searched on Google. `http://pypl.github.io`

Why Python?

Python has become the standard for deep learning practitioners, with R being the language of choice for statisticians.



source: indeed.com (2017)

The 4C16 Lab System

Overview: Main Components

Under the hood, the 4C16 lab system is a fairly complex, there are 4 components that you will need to be familiar with:

- the `dashboard` at `http://turing.mee.tcd.ie/`, where you can manage your account and monitor your progress.
- the `git server` on turing.mee.tcd.ie, which hosts:
  - a master repository to backs up your files
  - an assessment repository to grade your labs
- your personal `google drive`, which allows you to access files, and configuration settings when running a Colab machine.
- `Colab`, which will run a new virtual machine every time you start a colab notebook.

Overview: Main Steps

Here is a quick overview of the steps you will have to follow. More details instructions follow in the coming slides.

The first time around, you will need to do two steps:

1. Change default password on turing and get your private key.
2. Open `4c16-init.ipynb` in colab, paste the key, and run the cells to setup your google drive.

On daily usage, you will run in parallel 2 Colab notebooks:

1. `4c16-git-workflow.ipynb`, for fetching code on your google drive, pushing to turing, and submitting labs for grading.
2. the current lab's colab notebook instructions, found in your google drive, (eg. `4c16-labs/code/lab-00/Lab 0.ipynb`).

Initialisation — Detailed Instructions

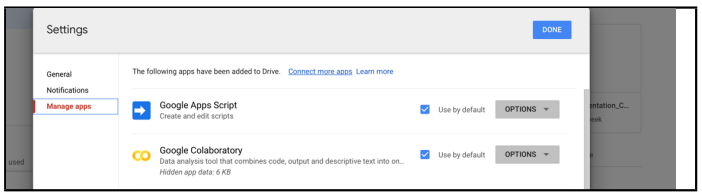
Step 0: Install Colaboratory

Adding Colab to your Google Drive

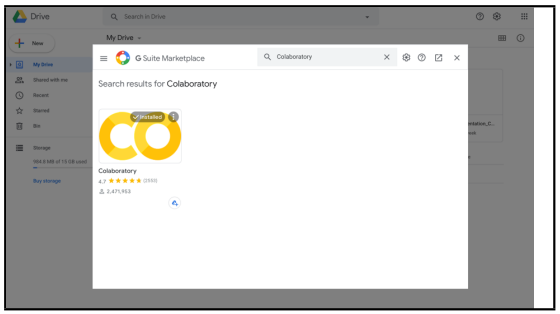
In case `4c16-init.ipynb` doesn't open automatically with colab, you will need to add Colaboratory application to your Google Drive:

Go to your Google Drive settings:

Select `Manage apps` > `Connect more apps`:



Search and install Colaboratory:



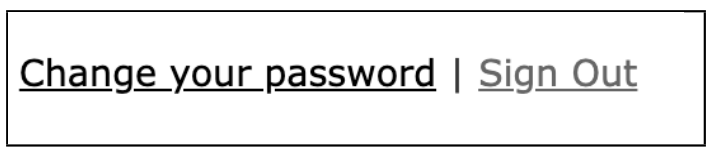
Step 1: `http://turing.mee.tcd.ie` (1/2)

- open a new browser
- go to `turing.mee.tcd.ie`
- login with your tcd email and password

Step 1: http://turing.mee.tcd.ie (2/2)

change your password!

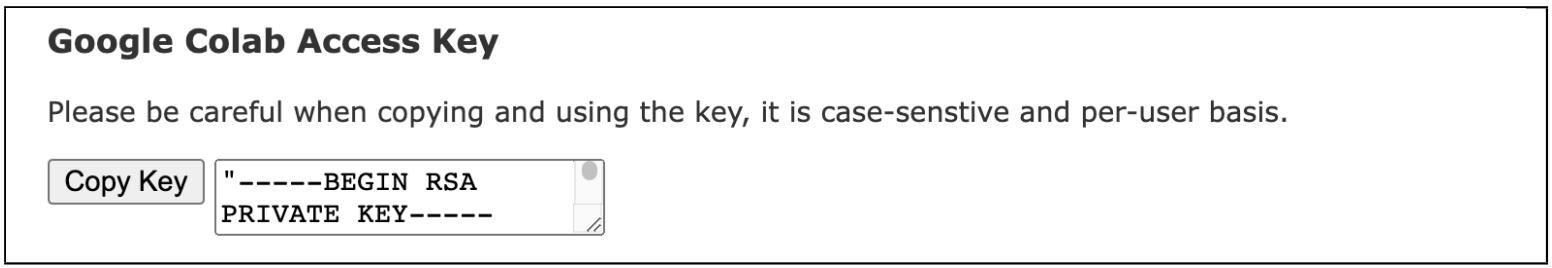
the link is at the bottom right of the page:



Step 1: http://turing.mee.tcd.ie (2/2)

Copy Private to clipboard

the link is here:



This private key will be used by the colab machines to communicate with turing.

Step 2: `4c16-init.ipynb` (1/4)

once-off installation: setting up your google drive

You need to know is that every time you open a new colab notebook, Google will spin a **new** machine for you. This machine knows has a fresh installation by Google, and knows nothing about your files, so it will need to communicate with something. The way we resolve this is by mounting your google drive in your colab runtime.

Your google drive is thus used to share lab material with your colab machines.

We must now prepare our google drive and install there a few things for our labs. This only needs to be done once at the beginning of term.

Step 2: `4c16-init.ipynb` (2/4)

once-off installation: setting up your google drive

On turing open (Init Notebook) link to the `4c16-init.ipynb` document.

For most of you this will be automatically open with colab, but you might have to add Colaboratory application to your Google Drive (see next slide)

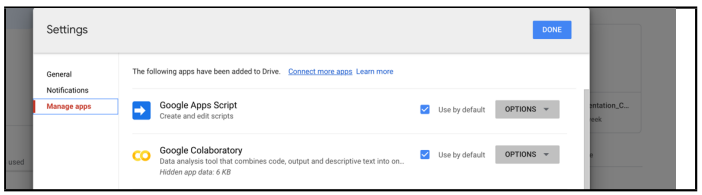
Step 2: `4c16-init.ipynb` (3/4)

Adding Colab to your Google Drive

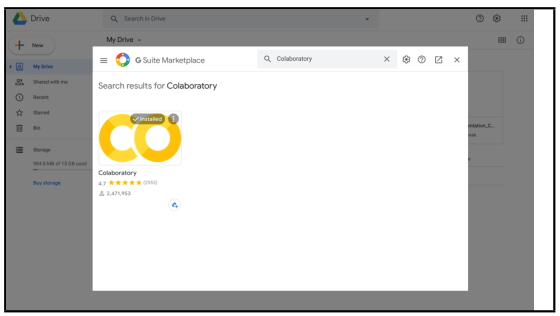
In case `4c16-init.ipynb` doesn't open automatically with colab, you will need to add Colaboratory application to your Google Drive:

Go to your Google Drive settings:

Select `Manage apps` > `Connect more apps`:

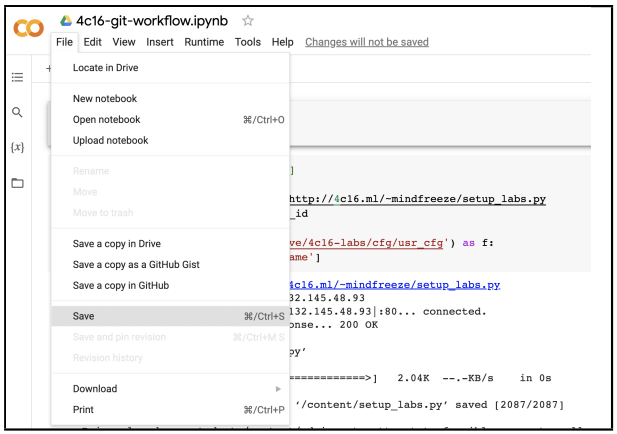


Search and install Colaboratory:



Step 2: `4c16-init.ipynb` (3/4)

This shared notebook is read-only, so you need to save a copy of the notebook on your Google Drive (default is to save it to the 'Colab Notebooks' folder).



Daily Usage — Detailed Instructions

4c16-git-workflow.ipynb (1/2)

daily git workflow

The lab system is reliant on a git server installed on `turing.mee.tcd.ie`. This git repository saves your lab files and also allows you to submit your labs and get an automatic evaluation.

Ideally we would be handling this with a terminal and use the command line commands such as `git clone`, `git commit`, `git push`, etc. (see brief git intro at end of slidedeck). But the free-tier version of colab doesn't feature a web terminal.

Thus, at each session, you will need to use the dedicated `4c16-git-workflow.ipynb notebook to manage your git workflow.`

4c16-git-workflow.ipynb (2/3)

daily git workflow

Open link to `4c16-git-workflow.ipynb` notebook provided to you on turing, make a copy to your google drive. You will be using this notebook to manage your daily git commands.

Note: this notebook must always be open before the lab's notebook.

This is because you need it to fetch a clean version of your lab material, which also include the lab notebook.

4c16-git-workflow.ipynb (2/3)

daily git workflow

We will make a clean clone of your lab directories at the start of every session. So if your edits that haven't been updated to the remote server on turing, they might simply disappear the next time you login (no kidding).

If you don't commit/push, your code doesn't exist!

(Couldn't write it bolder or redder, so you're on your own now).

4c16-git-workflow.ipynb (3/3)

The first two cells are used to checkout lab material in your google drive (you must always run this cell first, before starting working on a lab)

The notebook also contains a cell for you to save your current work (commit and push). Run this cell on a regular basis. Your work will be lost otherwise.

The last cell of notebook pushes submits your work to the grading system on turing. Read the output printout to see if you which tests your failed or passed.

Git Workflow Example

Getting Started with Lab 0

1. Open your `4c16-git-workflow.ipynb` notebook from your google drive.
2. Set `working_lab = 0` in first cell, evaluate next cell.
3. In google drive, go to `MyDrive/tcd-4c16-labs/code/lab-00/`
4. Open `Lab 0.iynb`, read it and follow the instructions.

- every time a chunk of work has been done, use `4c16-git-workflow.ipynb` to commit/push
- every time an exercise is done, submit lab for grading using `4c16-git-workflow.ipynb`

SCM

Git overview

It is essential to use source code management (version control) for any serious programming.

We use Git SCM system — but we will only use some of the features (after all, we are not working collaboratively) and all you will be using is packaged in the 4c16-git-workflow.ipynb notebook.

Here is a brief overview of what we use:

- Initialise a *repository* (or *repo*)
- Make some modifications to your code
- Review and modifications and decide which to *stage*
- *Commit* stage modifications
- *Push* to a repo (back up/collaborate)

Git workflow

The git workflow is to prepare a copy, or image, of your work that represents a step forward, and then *commit* that copy to the record. Example:

```
git add testfile.py
git status
git commit -m "[lab-00] added test file"
```

Here we have added the file testfile.py to the image for committing, checked that the repository is in the state we expect (via git status), and then committed our changes

Git workflow

After editing a file, you can examine the modification with

```
git diff testfile.py
```

We can also commit all modifications with

```
git commit -a -m "Updated plenty of files"
```

Update the remote (permanent backup) copy of your repo with:

```
git push
```

In our system, we have two remote called origin and assessment:

```
git push origin master
git push assesment master:students/{username}/code
```

Authors

Hugh Denman [Lab Infrastructure], PhD, adjunct professor at Trinity College Dublin, former engineer at Google and YouTube. His code is being used to transcode 400 hours of video/min in YouTube since 2012.

Vibhoothi [Colab Integration], Research Assistant, working on optimised transcoding with AV1 and other codecs. He is also part of various open-source initiatives and projects including Alliance For Open-Media (AOM), Xiph. Org, VideoLAN.

François Pitié [Labs], Assistant Professor in Media Signal Processing in the School of Engineering and part of the ADAPT Research Centre. His algorithms have been used by Google, Disney, Foundry, Weta Digital, and in many film productions.

FAQ

Q: Pushing for assessment outputs: `Everything up-to-date`
A: have you committed and pushed to master beforehand?