Module 2 : Open Source Collaborative Development

Session 2A : Spyder, Jupyter Notebooks and Google CoLab

Dr Daniel Chalk

"Beware the IDEs of March"

NIHR | Applied Research Collaboration South West Peninsula

@peninsula_ARC #HSMA4

HSMA 4

HSMA 4 Everyone

Module 2

Module 2 is a mini-module that runs parallel to Module 1. In Module 2, we'll talk about :

- the *tools* you'll use to code
- the principles of *Free and Open Source (FOSS)* software development
- *collaborative tools* and *version control*

In today's session, we're going to focus on the first of these – the *tools*.

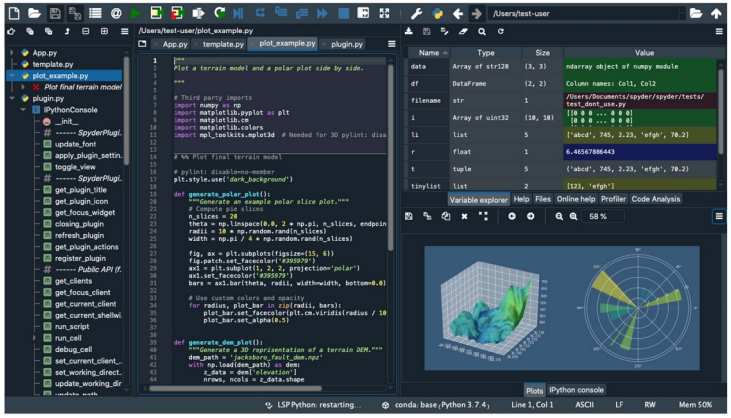
Integrated Development Environments

When you write computer code, it's strongly recommended that you use an *Integrated Development Environment*, or *"IDE"*

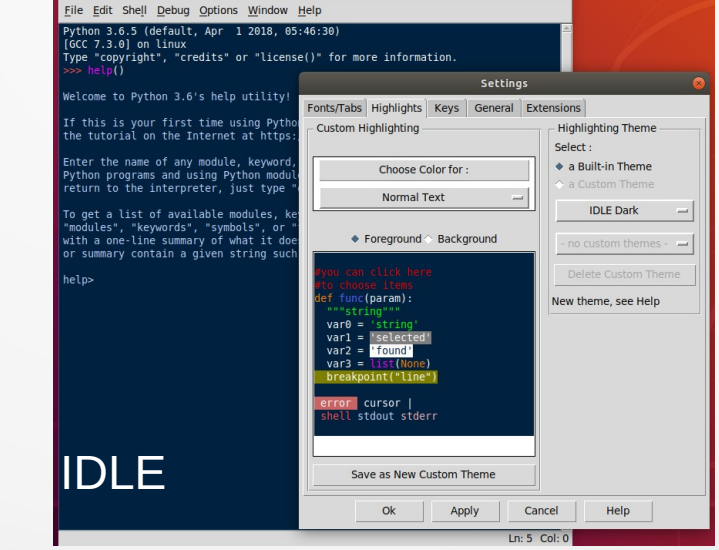
An IDE is a piece of software that provides tools that make coding a much more friendly process. This includes things like being able to debug your code (check for errors), interrogate your program "live", and automatically identify "syntax errors" (a bit like a spellchecker for coding) and other warnings as you write your code.

Some Popular IDEs for Python and R

SPYDER
The Scientific Python Development Environment



Microsoft
Visual Studio Code

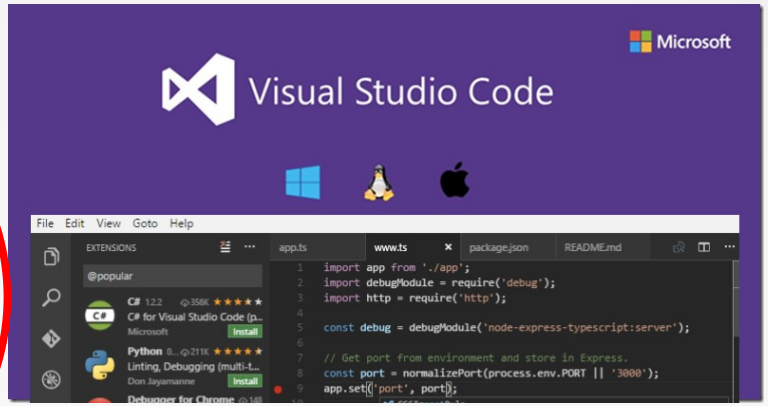


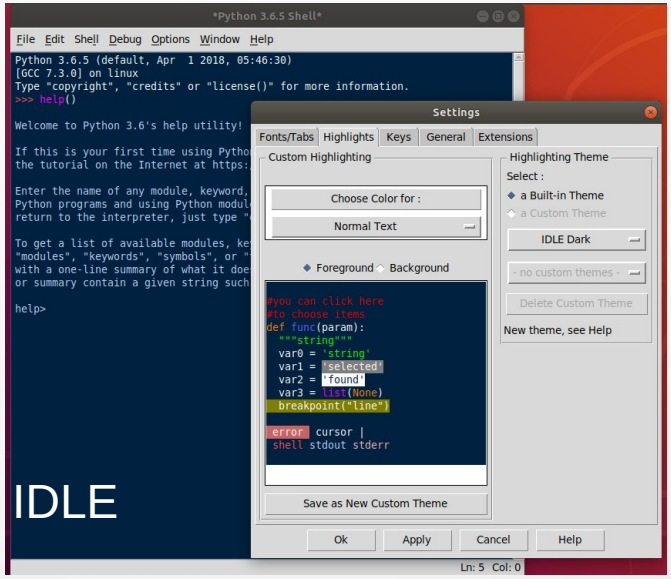
R Studio

jupyter

Vim

Some Popular IDEs for Python and R





Spyder

Spyder is a Free and Open Source IDE that we'll be using for most of the content on the course

It's an IDE designed by and for scientists, data scientists and engineers. It's included with the Anaconda distribution of Python.

It comes with a number of useful features. Let's have a look at it.

Spyder

Spyder (Python 3.7)

File Edit Search Source Run Debug Consoles Projects Tools View Help

/home/dan/Dropbox/HSMA_Onslaught/HSMA_4/HSMA_4_Phase_1_Training/1A_Introduction_to_Operational_Research_and_Data_Science

...SMA_4_Phase_1_Training/1A_Introduction_to_Operational_Research_and_Data_Science/how_long_spend_ed.py

how_long_spend_ed.py

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Oct 29 11:54:44 2019

@author: dan
"""

import simpy
import random

# Our patient arrivals generator
def arrival_generator(env, mean_interarrival_time, receptionist, triage_nurse,\
                      treatment_cubicle, mean_registration_time, \
                      mean_triage_time, mean_treatment_time):
    # Keep generating until the simulation finishes
    while True:
        # Create a new patient with the necessary attributes
        p = ed_patient(env, receptionist, triage_nurse, treatment_cubicle, \
                       mean_registration_time, mean_triage_time, \
                       mean_treatment_time)

        # Have the environment process the new patient (bring into being)
        env.process(p)

        # Randomly sample the time until the next patient arrives and have the
        # generator function timeout until then
        # We'll use an exponential distribution here
        t = random.expovariate(1.0 / mean_interarrival_time)

        # Activate the timeout
        yield env.timeout(t)

# The function that defines the journey of an ED patient
def ed_patient(env, receptionist, triage_nurse, treatment_cubicle, \
               mean_registration_time, mean_triage_time, mean_treatment_time):
    # Patient enters queue for receptionist
    # Record the time the patient entered the queue
    time_ent_q_for_recep = env.now

    # Record the time the patient entered the system
    time_ent_system = env.now

    # Request a receptionist resource and do these things only once one is
    # available
    with receptionist.request() as req:
        yield req

        # Record the time the patient left the queue for the receptionist
        time_left_q_for_recep = env.now

        # Calcuate the time spent in the queue for the receptionist and add
        # to the list of queuing times
        list_of_q_recep.append(time_left_q_for_recep - time_ent_q_for_recep)

        # Randomly sample time patient spends with receptionist
        time_with_recep = random.expovariate(1.0 / mean_registration_time)

        # Timeout the function until the patient has finished registration
        yield env.timeout(time_with_recep)
```

Name	Type	Size	Value
env	core.Environment	1	Environment object of simpy.core module
list_of_q_recep	list	66062	[0, 0.8325868757805652, 0.0, 0.0, 0.0, 0.0, 0.0, 2.1324871465686073, 0 ...
list_of_q_treat	list	66053	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...]
list_of_q_triage	list	66062	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...]
list_of_system_times	list	66049	[15.406302164037253, 17.51237785066318, 7.2635043734109495, 12.9288206 ...
mean_interarrival_time	int	1	8
mean_queuing_time_reception	float	1	0.6924988966847379

Variable explorer Help Plots Files

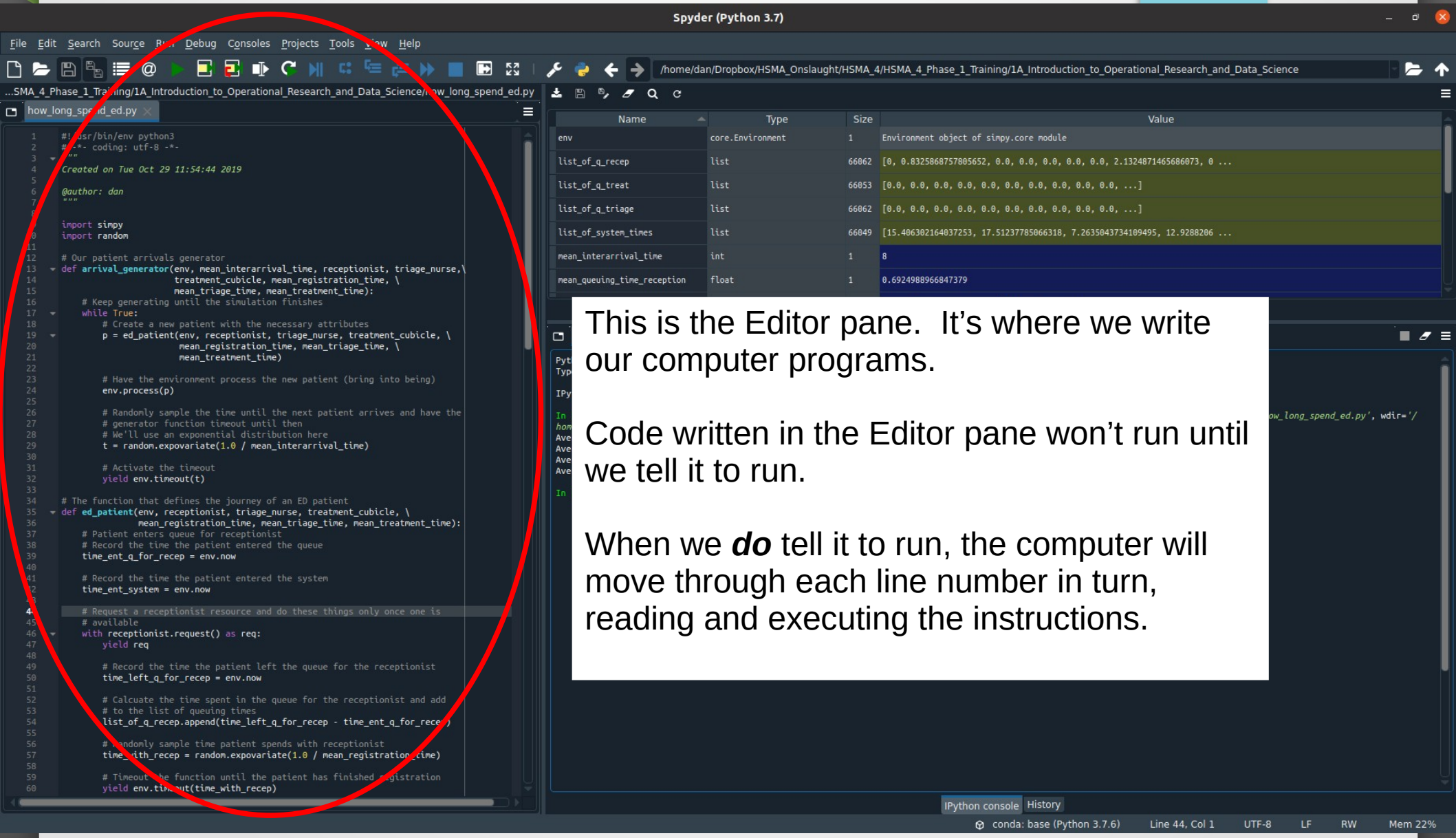
Console 1/A

```
Python 3.7.6 (default, Jan  8 2020, 19:59:22)
Type "copyright", "credits" or "license" for more information.

IPython 7.12.0 -- An enhanced Interactive Python.

In [1]: runfile('/home/dan/Dropbox/HSMA_Onslaught/HSMA_4/HSMA_4_Phase_1_Training/1A_Introduction_to_Operational_Research_and_Data_Science/how_long_spend_ed.py', wdir='/
home/dan/Dropbox/HSMA_Onslaught/HSMA_4/HSMA_4_Phase_1_Training/1A_Introduction_to_Operational_Research_and_Data_Science')
Average queuing time for reception : 0.69 minutes.
Average queuing time for triage    : 0.54 minutes.
Average queuing time for treatment : 127.44 minutes.
Average time in system : 165.65 minutes.

In [2]:
```

IPython console History

conda: base (Python 3.7.6) Line 44, Col 1 UTF-8 LF RW Mem 22%

Spyder



This is the Editor pane. It's where we write our computer programs.

Code written in the Editor pane won't run until we tell it to run.

When we ***do*** tell it to run, the computer will move through each line number in turn, reading and executing the instructions.

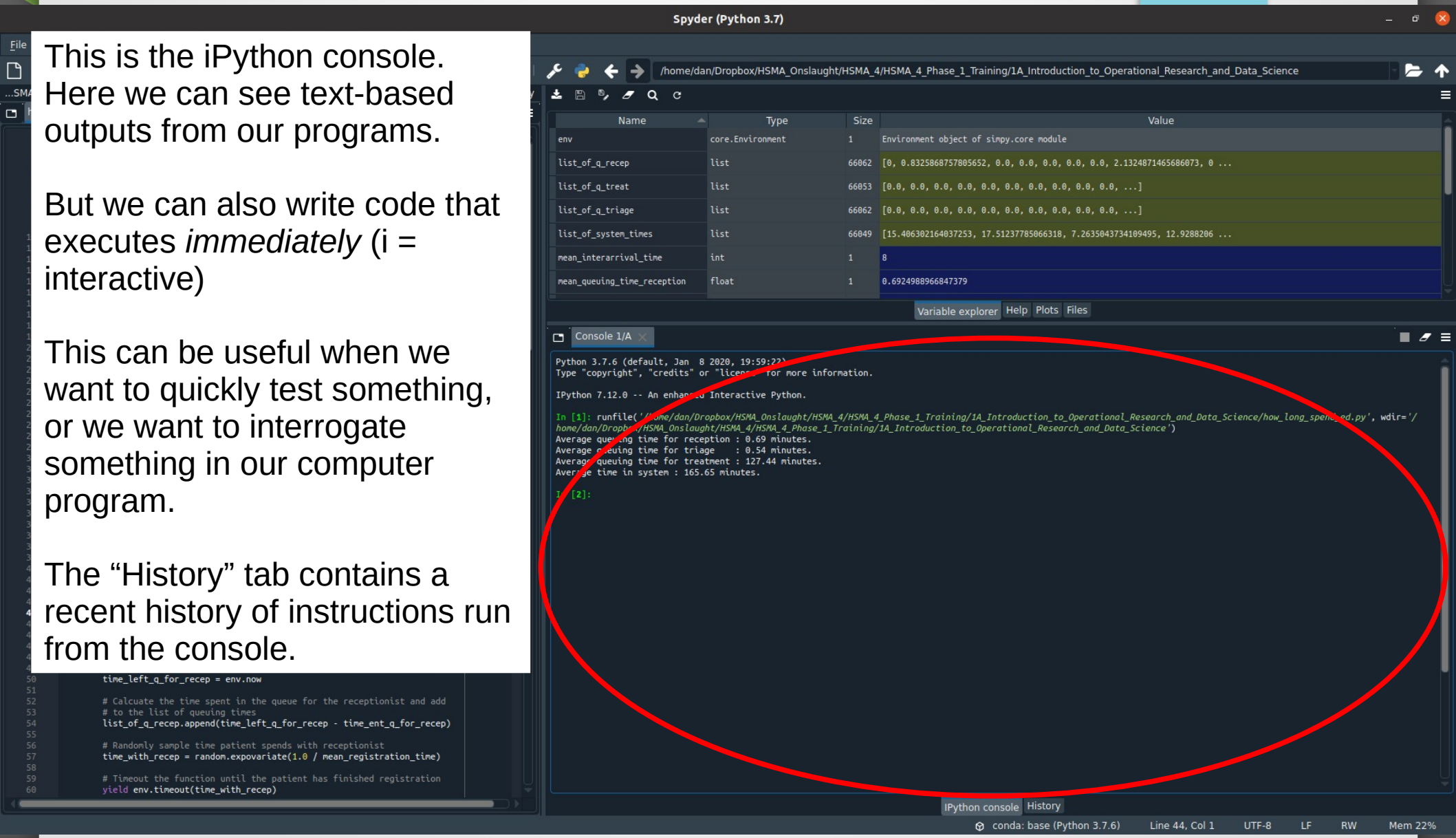
Spyder

This is the iPython console. Here we can see text-based outputs from our programs.

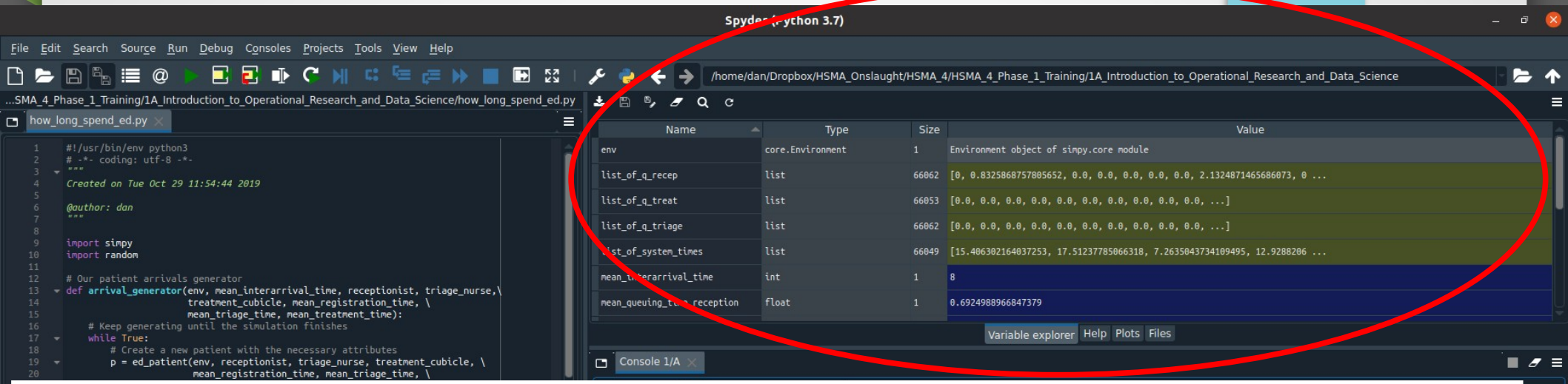
But we can also write code that executes *immediately* (i = interactive)

This can be useful when we want to quickly test something, or we want to interrogate something in our computer program.

The “History” tab contains a recent history of instructions run from the console.



Spyder



This pane contains a number of useful tabs.

The “Variable Explorer” allows us to see a list of all of the variables, and their current values, currently stored in memory.

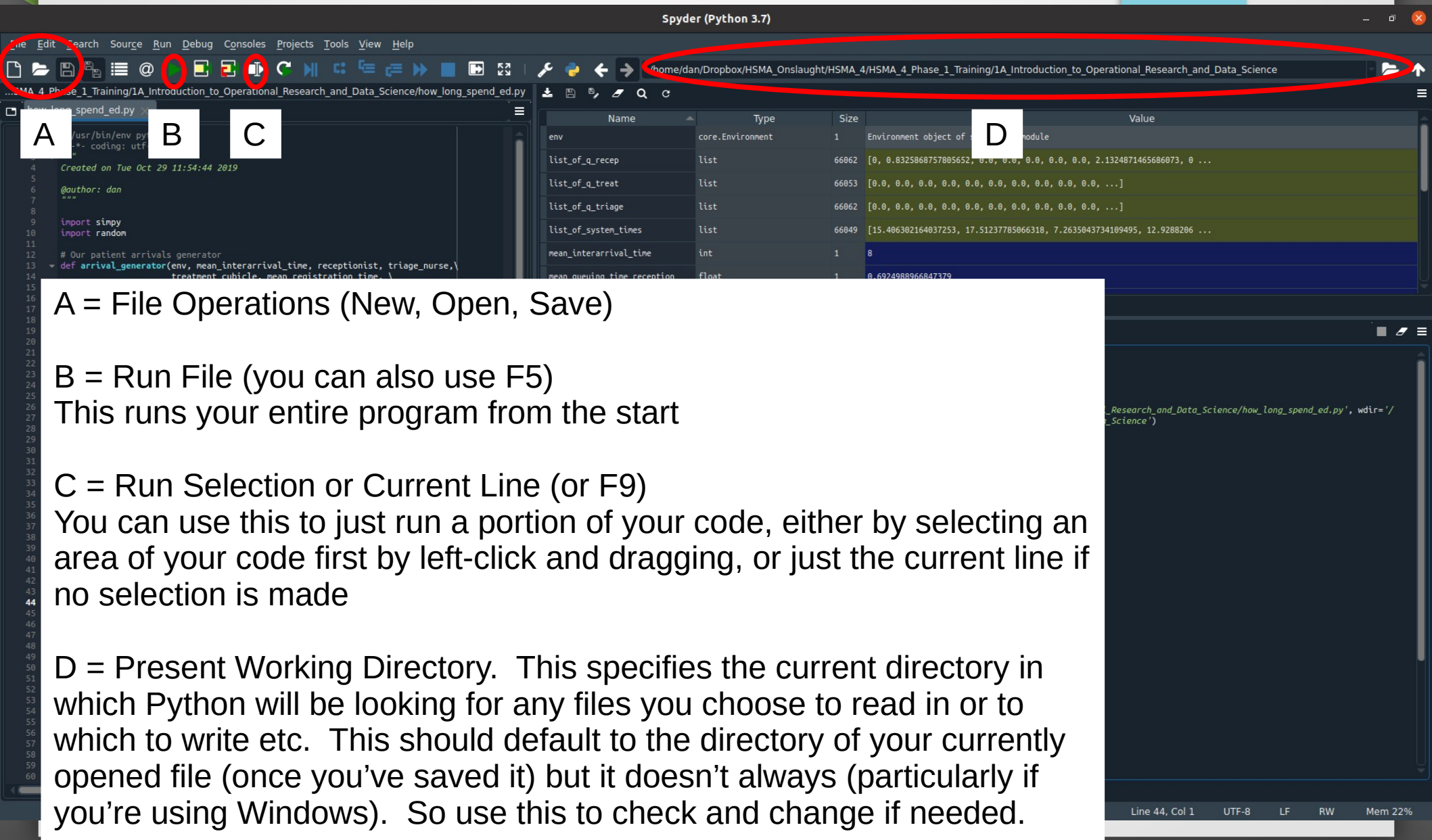
This can be enormously useful for making sure our code is working the way it should.

The “Plots” tab allows us to view graphs generated by our code – we’ll come on to that later in Module 1.

The “Help” tab allows us to push CTRL+I in front of an instruction, and get help documentation about it.

The “Files” tab allows us to explore, view and open files in our “Present Working Directory”

Spyder



A = File Operations (New, Open, Save)

B = Run File (you can also use F5)
This runs your entire program from the start

C = Run Selection or Current Line (or F9)
You can use this to just run a portion of your code, either by selecting an area of your code first by left-click and dragging, or just the current line if no selection is made

D = Present Working Directory. This specifies the current directory in which Python will be looking for any files you choose to read in or to which to write etc. This should default to the directory of your currently opened file (once you've saved it) but it doesn't always (particularly if you're using Windows). So use this to check and change if needed.

The Line

Did you notice the big vertical line running down the Editor pane?

This is a line which indicates a width of 80 characters.

Back in the old days, old terminal monitors had a standard width of 80 characters.

Back in the even older days, when programs were run on punch cards, and computers were bigger than your house, the IBM punch card had 80 columns.

But even in the modern day, an 80 character width helps keep code readable.

Crossing The Line

It’s extremely bad practice to type over the 80 character limit line.

You can – Spyder won’t stop you.

But you really shouldn’t.

I’ve heard it all :
“It’s only 1 character over the line, what does it matter?”
“I’ve got a 4K monitor and can see 600 columns if I want. Why should I care?”

It’s good to get in the practice of not crossing the line right from the off.

How Not to Cross the Line

So what do you do to avoid crossing the line?

Give all your variables really short but meaningless names?

```
m = a + (dp * c)
```

Only write very simple code?

```
answer = 2 + 2
```

Fortunately, there are much better ways. And we'll show you how in the Python training.

Exercise 1

You're now going to have a go at using Spyder! You need to work through the instructions in the file 2a_exercise_1.pdf.

You'll be put into breakout rooms in your Peer Support Groups. Each of you will need to work through the exercise, but you should talk to each other as you do, to share ideas, offer / ask for help when people are stuck etc. You should undertake the last question together as a group.

You've got 40 minutes.

Jupyter Notebooks

Jupyter Notebooks are a type of *document*, much like a Word document (.docx) or a Portable Document File (.pdf).

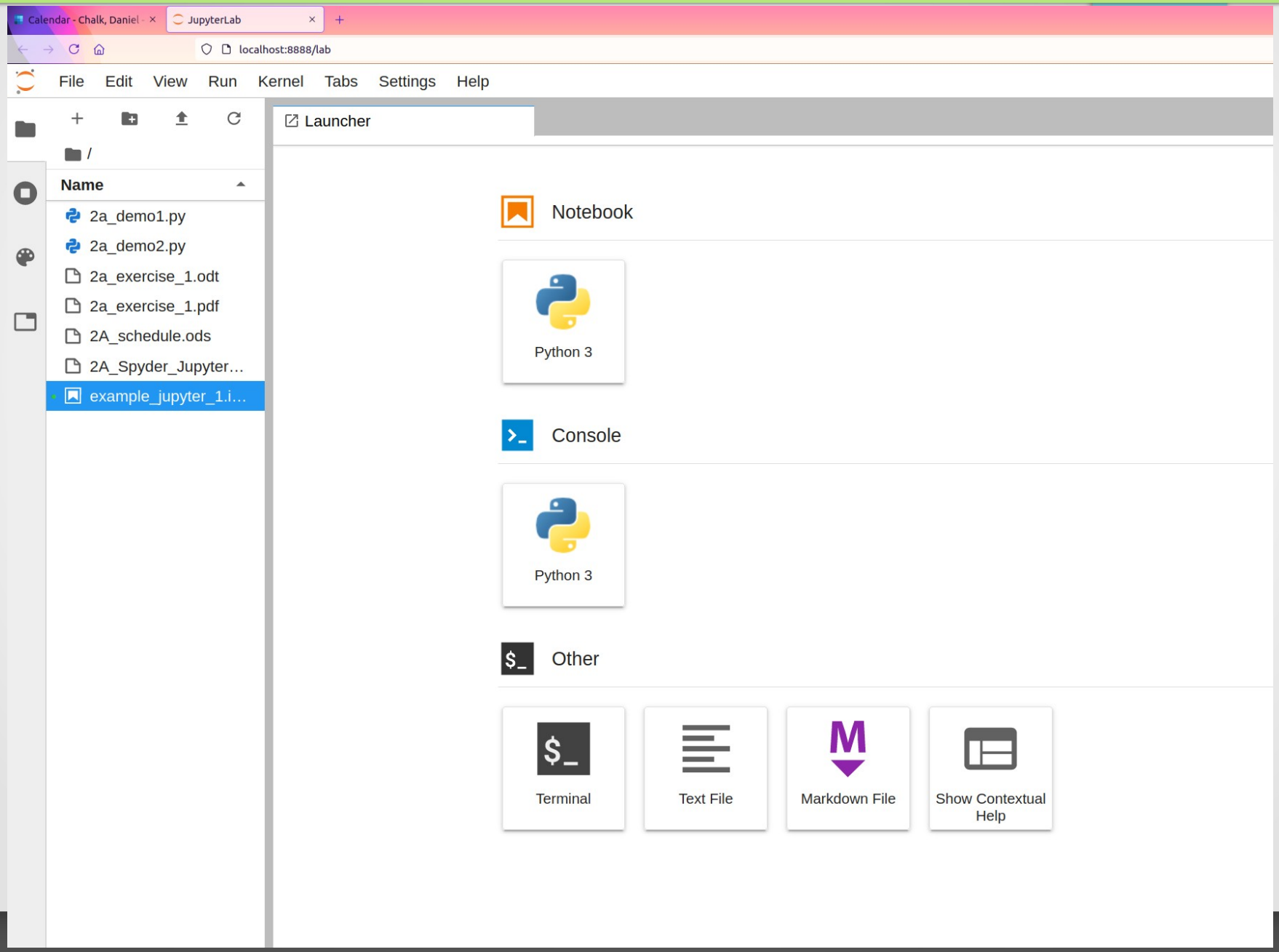
The key aspects of Jupyter Notebooks are :
- they are an *open* format
- they are made up of *cells*
- each cell can contain either narrative text / images (*markdown)* or a block of *Python code* that can be executed

Because of the above, they're great for sharing code (so you can explain what your code does in a friendly way), or for teaching!

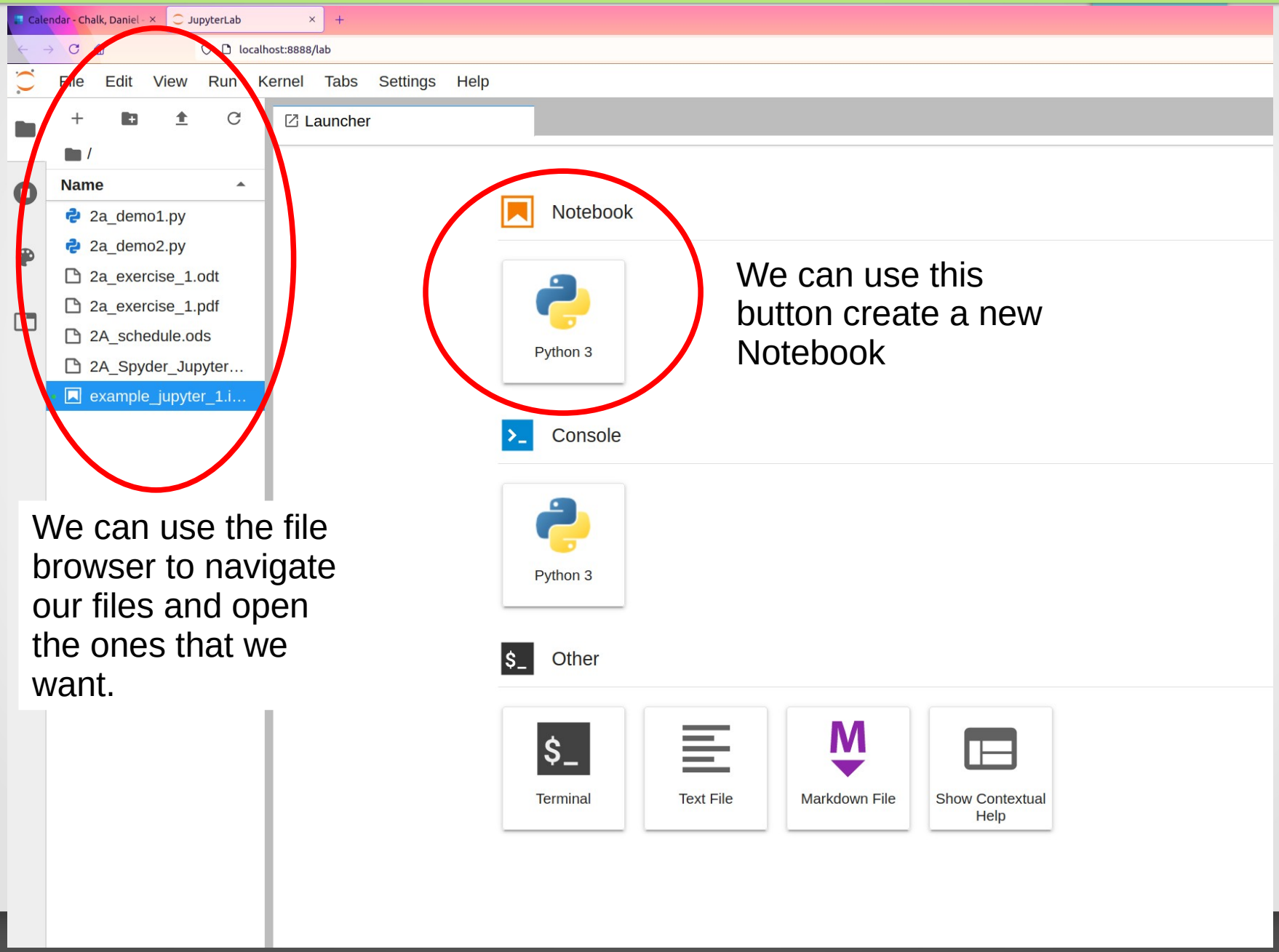
Let's have a look at them. We recommend using JupyterLab for working with Jupyter Notebooks.

Launch JupyterLab from Anaconda Navigator.

JupyterLab


Calendar - Chalk, Daniel
JupyterLab
localhost:8888/lab
File
Edit
View
Run
Kernel
Tabs
Settings
Help
Launcher
/
Name
2a_demo1.py
2a_demo2.py
2a_exercise_1.odt
2a_exercise_1.pdf
2A_schedule.ods
2A_Spyder_Jupyter…
example_jupyter_1.i…
Notebook
Python 3
Console
Python 3
Other
Terminal
Text File
Markdown File
Show Contextual Help

JupyterLab



We can use this button create a new Notebook

We can use the file browser to navigate our files and open the ones that we want.

Exercise 2

We'll first take a 10 minute comfort break. Upon your return, you'll work in your groups (as you did earlier) to do the following :

1. Open example_jupyter_1.ipynb from within Jupyter Lab and work through it.
2. Once you've done that, and you understand everything therein, have a go at creating your own Jupyter Notebook. Specifically, I want you to take the little bits of simple code you used in the first exercise from earlier, and put them into a notebook wrapped up with some explanatory text and appropriate markdown. Be creative!

You have 40 minutes for the exercise (+ 10 minute break now).

Important – in some cases, you may not be able to access Jupyter Lab due to firewall restrictions imposed by your organisation. If this is the case, you should ask your IT department to fix this for you, but for now, find someone in your group who can use it and suggest they share their screen for the whole group.

Google CoLab

Google CoLaboratory (CoLab) is an online platform run by Google that allows you to write and execute Python code in your browser.

You do not need to have any other software (such as Python or any IDEs) installed to use CoLab. This makes it a great way to share code with people who don't have such software installed.

Your code runs on CPUs on Google's servers, rather than locally on your own computer (as is the case with Spyder and Jupyter Notebooks). You also have access to GPUs and TPUs (Tensor Processing Units) – both of which are excellent for machine learning applications.

There are limits to how you can use CoLab with a free account – primarily, this relates to the priority you get for the CPU, GPU and TPU resources on Google's servers.

Google CoLab Notebooks

CoLab uses a Notebook-style format that is very similar to Jupyter Notebooks. In fact, you can even import Jupyter Notebooks directly into Google CoLab!

But CoLab does also have some unique features of its own.

Google CoLab is available here :
https://colab.research.google.com/

Google CoLab

Examples

Upload

This tab contains example notebooks that serve as guides for CoLab's features. They do assume that you know Python though, so I'd recommend taking a look at them once you've been through the Python training.

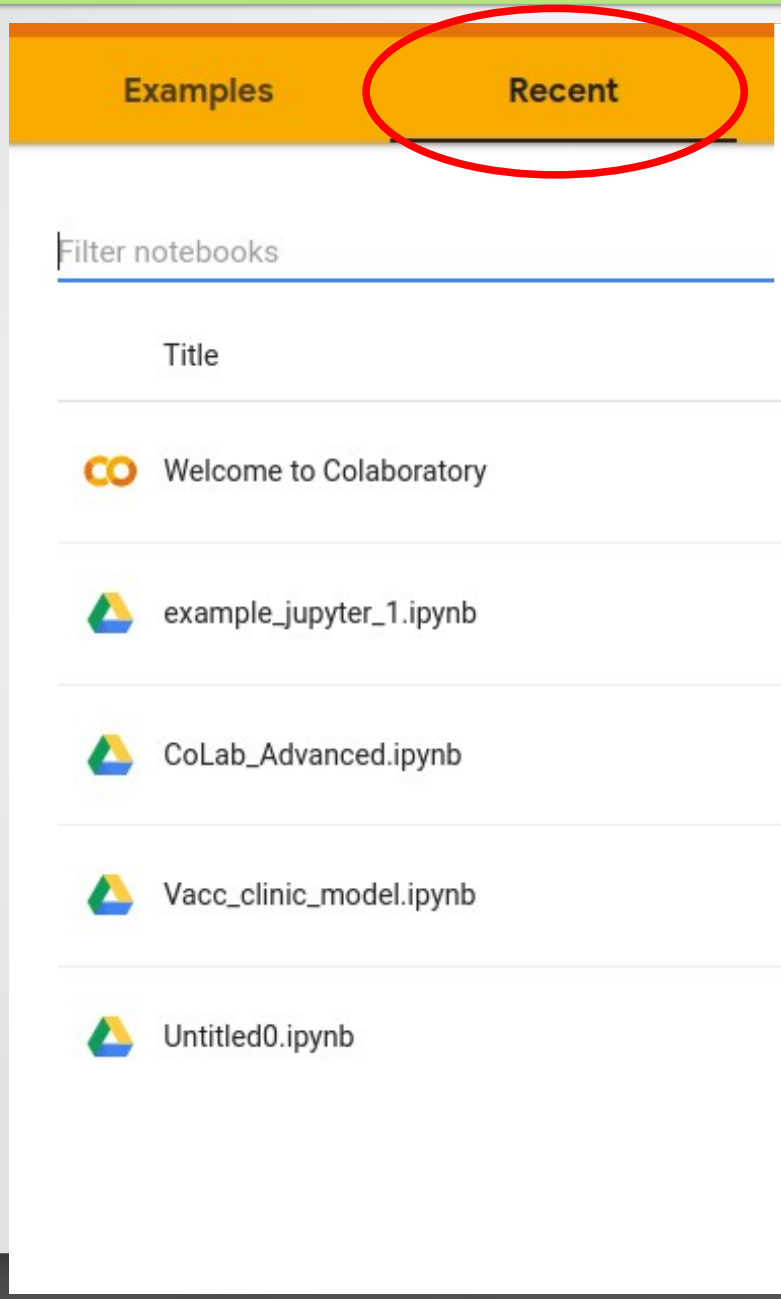
Filter notebooks

Title

- Overview of Colaboratory Features
- Markdown Guide
- Charts in Colaboratory
- External data: Drive, Sheets, and Cloud Storage
- Getting started with BigQuery

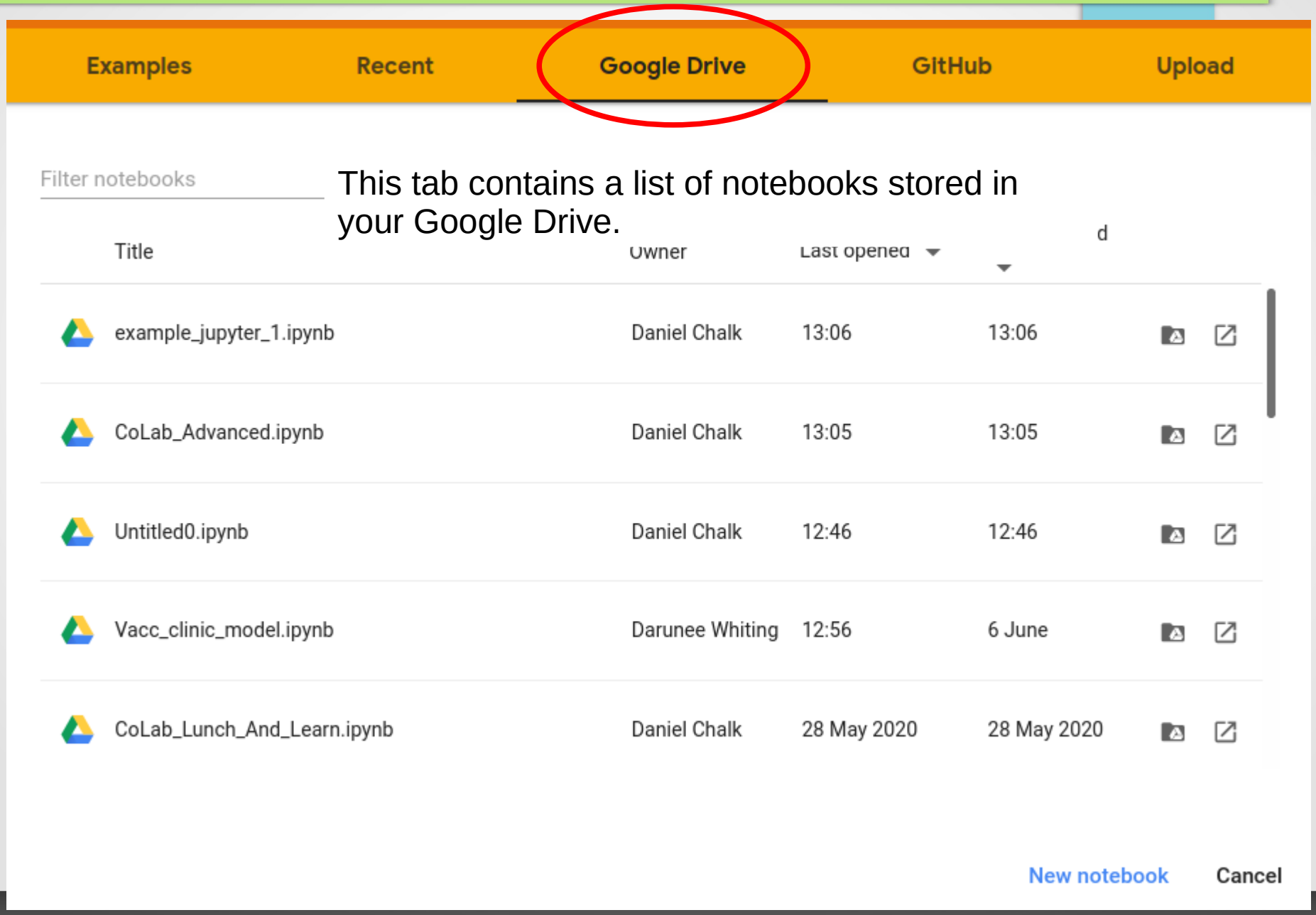
New notebook Cancel

Google CoLab



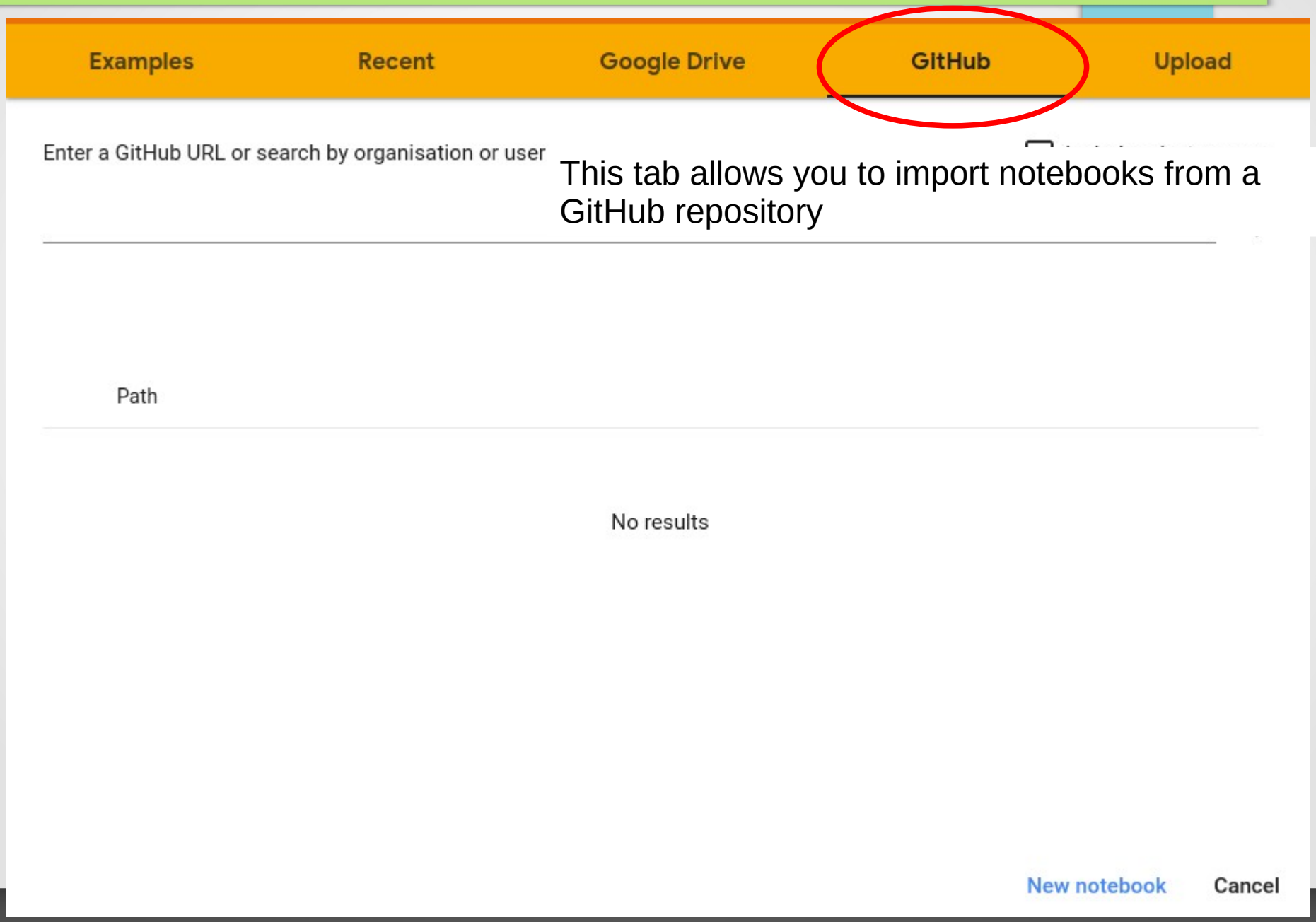
This tab contains a convenient list of your most recent files you've been working on / accessed in CoLab. If this is your first time using it, you'll just have the "Welcome to Colaboratory" notebook.

Google CoLab



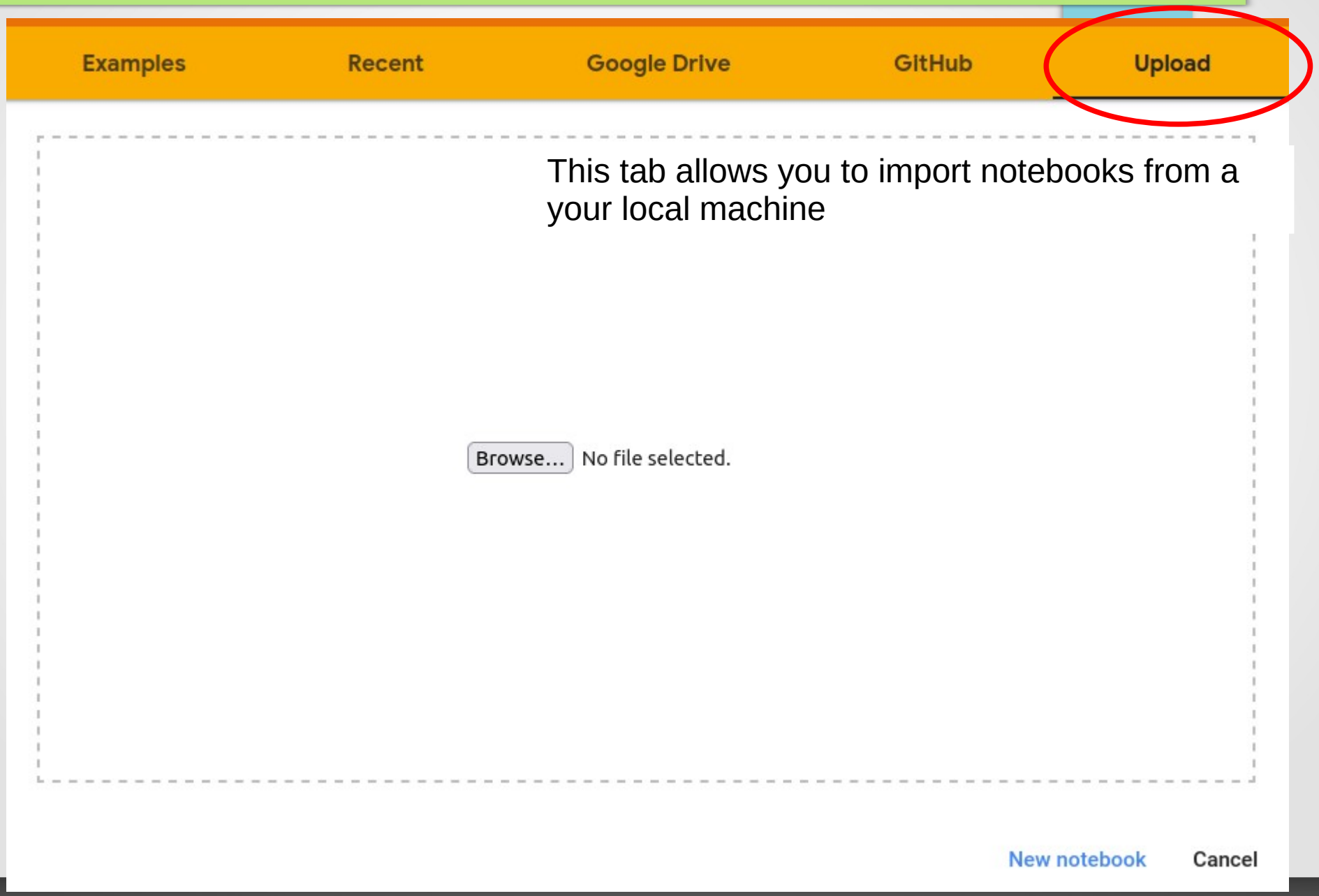
This tab contains a list of notebooks stored in your Google Drive.

Google CoLab



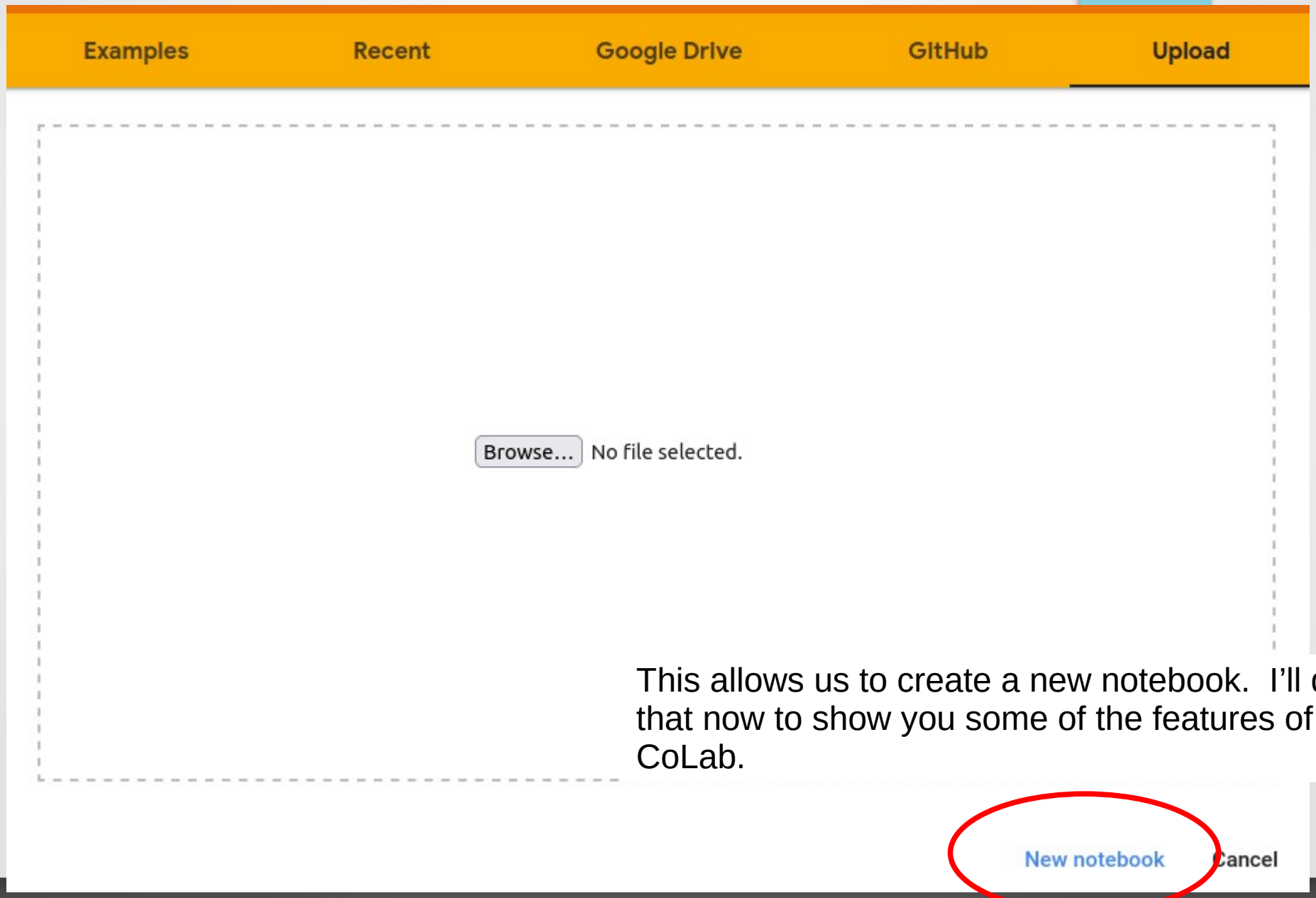
This tab allows you to import notebooks from a GitHub repository

Google CoLab



This tab allows you to import notebooks from a your local machine

Google CoLab



This allows us to create a new notebook. I'll do that now to show you some of the features of CoLab.

Overview of Basic Features

Here are some of the basic features we just looked at :

- Adding code and text cells
- Formatting using the cell toolbar (although markdown still works)
- Clicking off text cells to finalise them
- Clicking the play button to run code cells (or using CTRL + Enter)
- Clearing all outputs without restarting runtime
- Restarting runtime
- Expanding and collapsing sections using headers
- Runtime and changing hardware accelerator

Exercise 3

For your final exercise, you'll once again work in your groups. I want you to :

1. Read through the CoLab notebook I've written about some of the more advanced features of CoLab here : https://bit.ly/3w0JAuk
Try running the code in each cell after reading how it works, and try uploading and downloading some files. Talk about it in your groups.

2. Have a look at 2 Free and Open Source models shared online using CoLab :
   a) https://bit.ly/3hiInsU is a generic model that allows users to predict waiting times and capacity breaches for a vaccination clinic. This model was built by one of our HSMA 3 students – Dr Adam Kwiatkowski, a GP from North Devon
   b) https://bit.ly/3x2Ox78 is a model I developed in 2020 to predict the "stuff" and "staff" resourcing needs for end of life care during the pandemic.

Have a read and a play with both of these models, and chat about them in your groups. You're not expected to understand any of the code at this stage, but both models have been designed for non-experts to use. We'll come back at 12.25 for a final chat.