acadigia
Puts the digital in academia

Compendium of Augmented Blended Teaching & Learning for Open Pedagogic Tools

Blended Learning Tool Evaluation – Jupyter Notebook

w w w . a c a d i g i a . e u

Co-funded by the Erasmus+ Programme of the European Union

Blended Learning Tool Evaluation – Jupyter Notebook

acadigia
Puts the digital in academia

Co-funded by the Erasmus+ Programme of the European Union

Table of Contents

Co-funded by the
Erasmus+ Programme
of the European Union

Report Overview

This document describes Jupyter Notebook as a tool to support blended learning. Following testing and evaluation, it discusses the functionalities and how the tool supports educators in their professional activities, namely:

- Professional Engagement
- Digital Resources
- Teaching & Learning
- Assessment
- Empowerment of Learners
- Facilitating Learners' Digital Competence

1. Introduction

Jupyter notebook need to be considered as part of a broader framework, that of Project Jupyter. Project Jupyter is three things: a collection of standards, a community, and a set of software tools. Project Jupyter is a broad collaboration that develops open-source tools for interactive and exploratory computing. The tools includes: over 100 computer languages (with a focus on Python), the Jupyter Notebook, JupyterHub, and an ecosystem of extensions contributed by a large community. At the same time, the name “Jupyter" is a loose acronym meaning Julia, Python, and R. These programming languages were the first target languages of the Jupyter application, but nowadays, the notebook technology also supports many other languages.

Jupyter Notebook, the part of Jupyter we are concerned, is a free, open-source, interactive web tool software that creates a Jupyter notebook. A **Jupyter notebook is a document that supports mixing executable code, equations, visualizations, and narrative text**. This marriage of content and code makes for a powerful **new form of data-based communication.** Specifically, Jupyter notebooks allows the user to bring together data, code, and prose, to tell an interactive, **computational story**. Whether analysing a corpus of American Literature, creating music and art, or illustrating the engineering concepts behind Digital Signal Processing, the notebooks can **combine explanations traditionally found in textbooks with *the interactivity of an application*.**

The Jupyter Notebook has exploded in popularity since late 2014, fuelled by its adoption as the favourite environment for doing data science. It has also grown as a platform to use in the classroom, to develop teaching materials, to share lessons and tutorials, and to create computational stories. Educators everywhere are adopting Jupyter for teaching. This resource can be very useful for any educator teaching a topic that includes data analysis or computation in order to support learning. It is not just for educators teaching courses in engineering or science, but also data journalism, business and quantitative economics, data-based decision sciences and policy, quantitative health sciences, and digital humanities.

The main **components** of the whole Jupyter environment are, on the one hand, the **notebooks** themselves and the **application**. On the other hand, you also have a notebook **kernel** and a notebook **dashboard**.

The **web application** is a browser-based tool for interactive authoring of documents which combine explanatory text, mathematics, computations and their rich media output. As a “**web application**”, in which you can create and share documents that contain live code, equations, visualizations as well as text, the Jupyter Notebook is one of the ideal tools to help you to gain the data science skills you need. As a **server-client application**, the Jupyter Notebook App allows you to edit and run your notebooks via a web browser. The application can be executed on a PC without Internet access, or it can be installed on a remote server, where you can access it through the Internet.

As a "**notebook**" or "**notebook documents**" it consists of documents that contain both code and rich text elements, such as figures, links, equations, ... Because of the mix of code and text elements, these documents are the ideal place to bring together an analysis description, and its results, as well as, they can be executed perform the data analysis in real time. The Jupyter Notebook App produces these documents.

A **kernel** is a program that runs and introspects the user’s code. The Jupyter Notebook App has, by default, a kernel for Python code, but there are also kernels available for other programming languages.

acadigia
Puts the digital in academia

Co-funded by the Erasmus+ Programme of the European Union

The **dashboard** of the application not only shows you the notebook documents that you have made and can reopen but can also be used to manage the kernels: you can which ones are running and shut them down if necessary.

2. Testing and evaluation of the tool

The tool has been tested by doing an in depth desk based research using resources available on the web.

No physical implementation of the SW has been done because there were enough demo and examples of implementation available on the web page of the product and other web pages and no installation of the SW was needed at this stage of the testing. Installation is difficult to perform and requires time and effort that does not justify the endeavour at this moment of the research. Additionally, although the tool is very easy to use for the students, it is much more demanding for the teachers. The teachers needs to have programming experience in order to create a meaningful, full and attractive notebook with Jupyter.

Project Jupiter web page (https://jupyter.org/) provides all the information and demos required to get acquaintance of the product and its functionalities.

3. Functionalities supporting blended learning

3.1 Supporting Professional Engagement

- **Usage by a large number of students** is not a problem since this educational projects are available without needing to install commercial software. Moreover, they can be accessed through different devices, such as smartphones or tablets

- **Safety and security**. Jupyter can be run without network access avoiding any worries about security, as far as your computer is safe.

- **Local installation on students' or lab computers**. "Local installation" means that each computer is running the software that includes the Jupyter Notebook. Typically, this requires installing a distribution that includes Jupyter, Python, and possibly other language kernels. A popular software distribution that includes Jupyter is "Anaconda", which is easy to install on Windows, Mac, and Linux. Because it can install everything with user level permissions, it does not require the user to have administrator (or root) access to the computer. Two other easily installable software packages that can run Jupyter notebooks are "nteract" and "Hydrogen

  Teachers can ask students to install Jupyter on their own computer or make it possible for them to use it on lab computers. These can also be combined: give students the instructions to install it on their own, but also tell them that it's available in the lab if they can't get it to work on their laptop. This way you don't need a large enough computer lab for everyone, and don't need to worry that not everyone can get it to work on their own.

- **Jupyter on remote servers**. Even when Jupyter runs locally, it runs as a web application; that is, it runs in a browser connected to a server. In a local installation, the browser and the server run on the same machine. But it is also possible to run the server remotely. In that case, students don't have to install anything; they only have to run a browser and load a URL.

- **Initial barriers on local installations**. Because Jupyter can be installed by the students in their own computer, professors can adopt Jupyter without support or resources from their institution. However, this approach is only possible if every student owns a computer with enough capacity.

  Nevertheless, the starting process can become complicated. Although installation is generally easy, it still takes time. The time spend at the beginning of a class can be worthwhile for a semester-long course that uses Jupyter throughout, but it is a barrier to using Jupyter for a single module or one-off assignment in a course about something else.

  Also, the amount of time spent debugging problems scales with the number of students: a class of 25 students is bound to have a few people with 32-bit processors, incompatible libraries, out-of-date operating systems, over-zealous virus checkers, etc., and a class with 100 students will have four times as many. One work-around is to have students work in pairs: the probability that more than half of the students cannot get it working is reduced. Discrepancies in installed library versions can cause issues for students and may lead to different performances when students run code.

- **Operating systems:** Although Jupyter is cross-platform and ideally behaves the same on Windows, Mac, or Linux, and distributions such as Anaconda also behave very similarly on all platforms, the instructions for installing and launching it are slightly different on each operating system, so fine-grained instructions such as “double click here” or “type this command” need different versions for Linux, Mac, and Windows users, which can be challenging when the instructor presenting the material has only one platform at their disposal. It is worth developing detailed instructions that the students can go through at their own pace, rather than relying only on a live demo in class that will only apply to a fraction of the students.

3.2 Supporting Digital Resources

Jupyter notebook is so broad that educators newly adopting Jupyter can be overwhelmed by having to navigate the ecosystem of tools and content. Main digital resources in the Jupyter Notebook ecosystem are described here after.

Distribution and collection of materials. A variety of options are available for distributing course materials to and collecting them from students. Jupyter notebooks are plain text computer files, so you can distribute them to students and collect them using any system that handles text files, including GitHub, Google Drive, and (as a last resort) email attachment.

Learning management systems. Many instructors use a Learning Management System (LMS) to communicate with students. These tools offer private file sharing and assignments that connect to the students’ institutional computing accounts and they can be used to distribute and collect notebooks as text files. However, most LMS tools are not yet notebook-aware, so they don’t render notebooks or make it easy for instructors to comment on or grade them. Some tools and workflows are being actively developed to connect the Jupyter ecosystem to the LMS ecosystem using the Learning Tools Interoperability (LTI) standard.

Web hosting: Notebooks can be publicly hosted on any website, so students can download the files by clicking on a link. Most web-hosting software is not notebook-aware, but you can use nbviewer to

Co-funded by the Erasmus+ Programme of the European Union

share public notebooks, rendered as a static web page. nbviewer is a web service provided by Project Jupyter.

GitHub: One of the popular tools for distributing and collecting notebooks is GitHub, a hosting and collaboration platform for software. GitHub is based on git, a version-control system. Educators at academic institutions can use GitHub Classroom, which allows instructors to set up assignments for a class. Students click on a link for an assignment and a copy of the assignment repository is created and initialized with the assignment content, which can be a notebook. Each student's repository can be made private, with access only granted for the student and instructor. This can be an efficient way to distribute assignments to a large class. A drawback of git is that it is hard to use. It might be worth spending time in your class to teach git, if it is valuable for students to learn about version control. But if this is not one of the learning goals for your class, you can minimize the students' exposure to git using graphical interfaces like GitHub Desktop and git for Windows. The default git tools for comparing files and merging changes do not work well with Jupyter notebooks. However, some specialized tools can help with these tasks .

JupyterHub: If students are using JupyterHub, teacher can place notebooks and any related files directly into the students' directories manually or via a script. If nbgrader is available on your JupyterHub instance you can use it to collect and distribute notebooks. This allows you to develop the notebooks and incrementally make them visible to the students for them to "fetch". They can then edit the notebooks or create new ones in the directory created in their storage space, and then publish their notebooks back to you for downloading, viewing, or assessing with the nbgrader tools

The **amount of online resources** devoted to Jupyter Notebook keeps increasing at an exponential rate. Thus, the following is not an exhaustive list of resources. Rather, it is a very short selection of these, whose validity may change during time. However, due to the relevance of these sites, it is likely that, even though new links will appear, the ones considered below will still stand out in the future.

Jupyter Notebook Documentation.

Reddit forum devoted to Jupyter Notebook.

An interesting collection of Jupyter Notebooks.

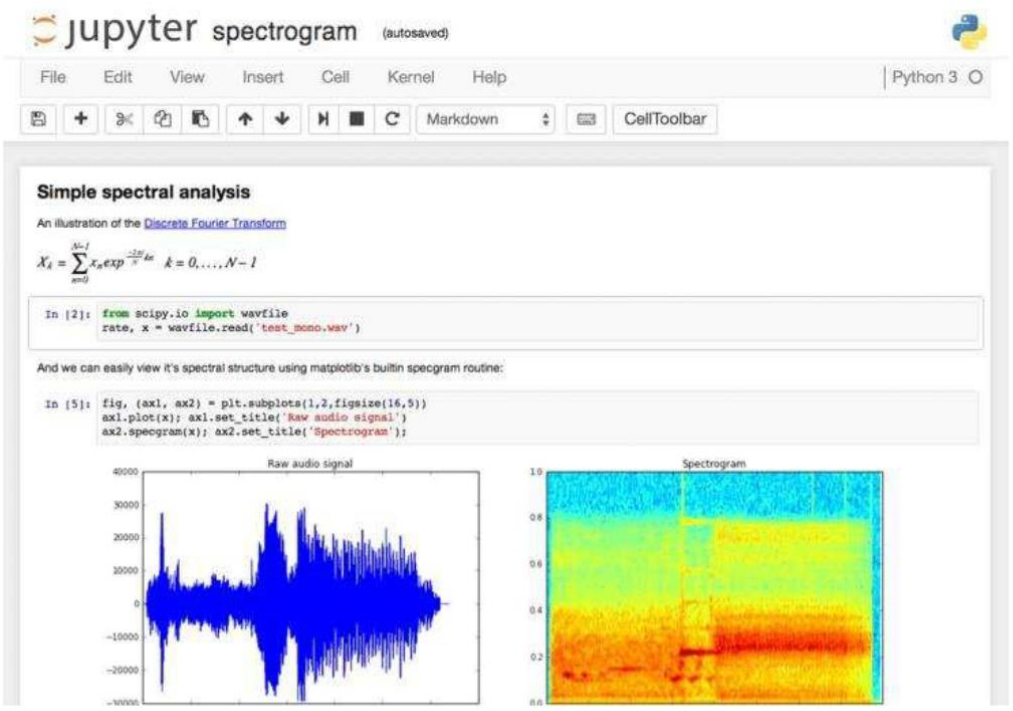
CoCalc's development webpage.

3.3 Supporting Teaching & Learning

Jupyter notebooks can be used to **organize classroom materials and objects**, **store and provide access to reading materials** for students, **present and share lecture** materials, **perform live coding**, explore and **interact with materials**, support **self-paced learning**, **grade** students' homework, **solve homework** problems, or **make materials reusable** to others.

Co-funded by the Erasmus+ Programme of the European Union

Next figure illustrates a Jupyter Notebook sample containing highlighted texts, mathematical formula, Python codes, images, which is used as lecture slides conveyed to students. Note that since code and notebook narratives are integrated into one page, students can easily practice the programming techniques they learned in class by writing and running the code within the page.


jupyter spectrogram (autosaved)
File Edit View Insert Cell Kernel Help
Python 3
Markdown
CellToolbar
Raw audio signal
Spectrogram

Simple spectral analysis

An illustration of the Discrete Fourier Transform

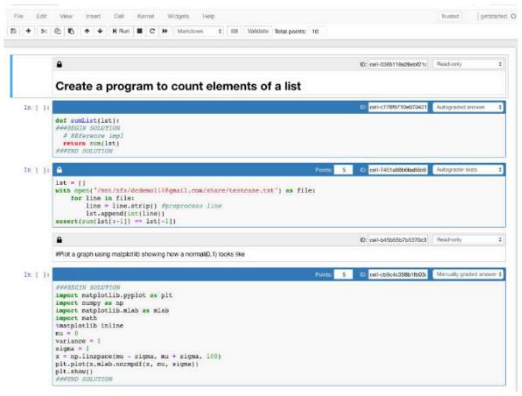
$$X_k = \sum_{n=0}^{N-1} x_n exp^{\frac{-2\pi i}{N} kn} \quad k = 0, \ldots, N-1$$

```
In [2]: from scipy.io import wavfile
        rate, x = wavfile.read('test_mono.wav')
```

And we can easily view it's spectral structure using matplotlib's builtin specgram routine:

```
In [5]: fig, (ax1, ax2) = plt.subplots(1,2,figsize(16,5))
        ax1.plot(x); ax1.set_title('Raw audio signal')
        ax2.specgram(x); ax2.set_title('Spectrogram');
```

During class, all the lecture notes, mathematical equations, figures, codes can be displayed through webpage, and students can execute the codes and results are immediately displayed in the same webpage which save teachers and students from frequently switching between slides and coding environment. After class, assignments can be created and the nBGrader function can automatically grade coding assignments if the input and output are clearly defined, and the centrality of the system makes answers collection and invigilation task of computer based exams much easier. The next figure shows how an assignment is created and released.



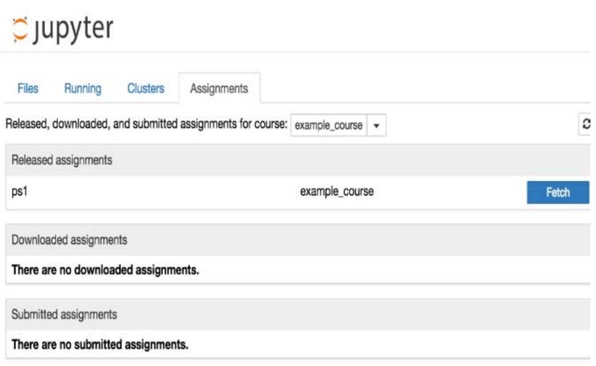
Create a program to count elements of a list

```
In [ ]: def sumList(lst):
        ###BEGIN SOLUTION
          # REference impl
          return sum(lst)
        ###END SOLUTION
```

```
In [ ]: lst = []
        with open("/mnt/nfs/dcdemo110@gmail.com/share/testcase.txt") as file:
            for line in file:
                line = line.strip() #preprocess line
                lst.append(int(line))
        assert(sum(lst[:-1]) == lst[-1])
```

#Plot a graph using matplotlib showing how a normal(0,1) looks like

```
In [ ]: ###BEGIN SOLUTION
        import matplotlib.pyplot as plt
        import numpy as np
        import matplotlib.mlab as mlab
        import math
        %matplotlib inline
        mu = 0
        variance = 1
        sigma = 1
        x = np.linspace(mu - sigma, mu + sigma, 100)
        plt.plot(x,mlab.normpdf(x, mu, sigma))
        plt.show()
        ###END SOLUTION
```



Released, downloaded, and submitted assignments for course: example_course

Released assignments

ps1	example_course	Fetch

Downloaded assignments

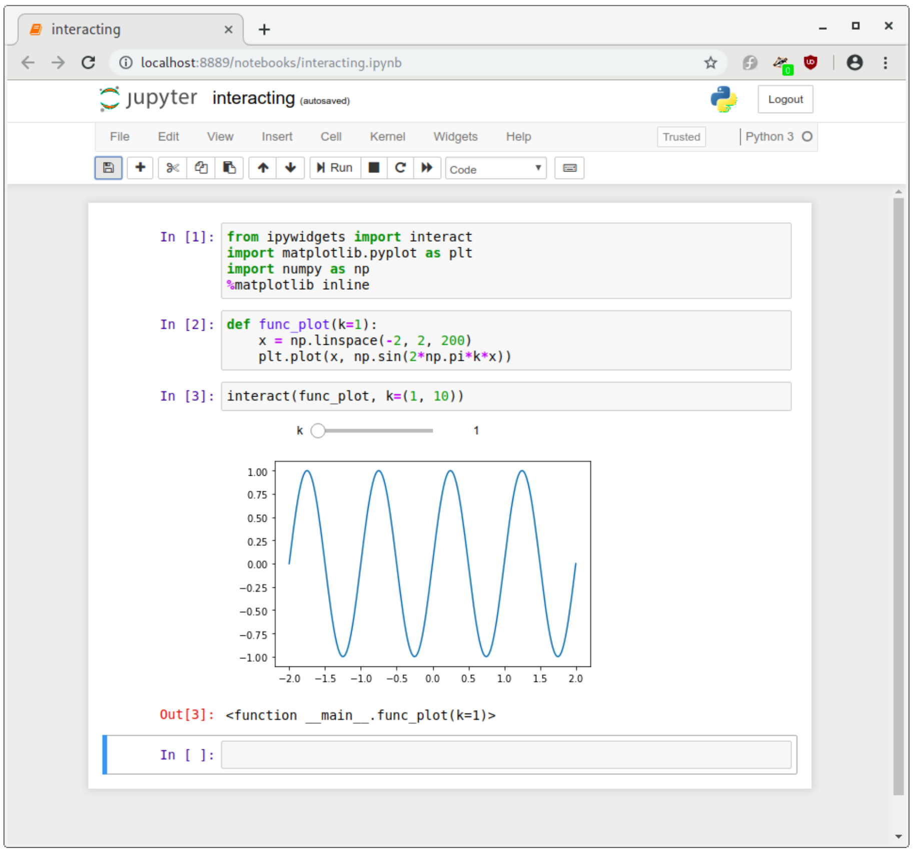
There are no downloaded assignments.

Submitted assignments

There are no submitted assignments.

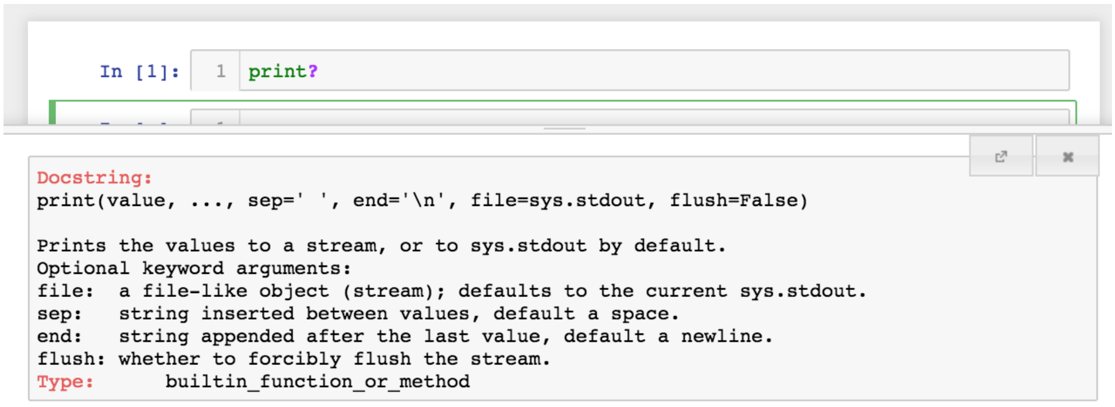
The first and most salient component of the notebook is the *cell*. The entire contents of a notebook is composed of only cells. These cells can take one **of two forms: text or code**. Code cells are composed of three areas: the **input** area, the **display** area, and the **output** area. The input area is identified by the In []: prompt to the left of the cell. Between the brackets of the In prompt can be one of three items: a number, an asterisk, or a blank. A number indicates that this cell has been executed and the value of the number indicates the order of execution.

Widgets. Widgets provide the opportunity for learners and instructors to interact with code outputs, such as charts and tables. Widgets are “mini” Graphical User Interfaces (GUI) that give the notebook user access to slide bars, toggle buttons, and text-boxes. They can be used in conjunction with code, allowing a change of mindset from programming as a primary goal to exploring a model or computation as the primary goal. Alternatively, the code can be hidden and left only the widgets used to create a notebook “app” that might connect input parameters with a simulation and a plot.


interacting
localhost:8889/notebooks/interacting.ipynb
jupyter interacting (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
k
1

```
In [1]: from ipywidgets import interact
        import matplotlib.pyplot as plt
        import numpy as np
        %matplotlib inline

In [2]: def func_plot(k=1):
            x = np.linspace(-2, 2, 200)
            plt.plot(x, np.sin(2*np.pi*k*x))

In [3]: interact(func_plot, k=(1, 10))

Out[3]: <function __main__.func_plot(k=1)>

In [ ]:
```

Accessing documentation in the notebook. From a notebook cell, the TAB key autocompletes and SHIFT-TAB brings up full documentation. Similarly, using a question-mark after a method or function will bring up the documentation after the cell is run. Using this feature in class during live coding or while explaining how code works helps make students comfortable of working effectively with libraries



```
In [1]:   print?
```

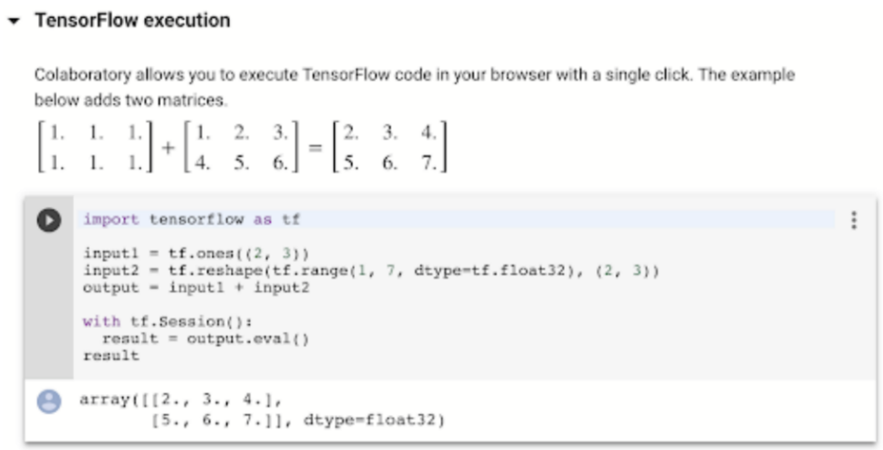
```
Docstring:
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
file:  a file-like object (stream); defaults to the current sys.stdout.
sep:   string inserted between values, default a space.
end:   string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.
Type:      builtin_function_or_method
```

Magics. Magics are meta-commands that only function within Jupyter and allow a user to access language/kernel-specific features. For instance, the IPython kernel provides a number of magics that can be useful while developing Jupyter notebooks using Python as the primary language. Many other magics are available for different kernels but they are specific to Jupyter so may not be usable in a stand-alone script in that language outside of Jupyter.

Notebooks under version control: Keeping notebooks under version control is a great way to not only keep track of changes to your content, but also for sharing it. In a course where multiple people are contributing to the development of notebooks for the course, using version control in conjunction with a platform like GitHub, allows authorship to be tracked and provides communication tools for reviewing new contributions or outlining requested development for a new assignment, activity, etc. Another advantage of using version control is that some services will provide rendered views of notebooks that you have made public. GitHub shows a rendered version of the notebook, rather than the ASCII text that a notebook is comprised of. Some pitfalls with LaTeX rendering may occur, as platforms do not always render the notebooks the same as they would appear in an active Jupyter interface.

Testing notebooks: Before distributing notebooks, at a minimum, you can test that the notebook executes cleanly from top to bottom by restarting the kernel and running all cells from top to bottom. This can easily be done from the menu (Restart + Run all).

Advanced topic: extensions. There are many communities contributed extensions that add functionality to Jupyter notebooks. Extensions vary from displaying an automated table of contents for a notebook, or prettify code, or hiding/showing solution cells. Next figure shows shows how Google Collaboratory, one of many tools to interact with Jupyter notebooks, leverages the power of Jupyter extensions for custom interaction and presentation.


▾ TensorFlow execution

Colaboratory allows you to execute TensorFlow code in your browser with a single click. The example below adds two matrices.

$$\begin{bmatrix}1. & 1. & 1.\\ 1. & 1. & 1.\end{bmatrix} + \begin{bmatrix}1. & 2. & 3.\\ 4. & 5. & 6.\end{bmatrix} = \begin{bmatrix}2. & 3. & 4.\\ 5. & 6. & 7.\end{bmatrix}$$

```
import tensorflow as tf

input1 = tf.ones((2, 3))
input2 = tf.reshape(tf.range(1, 7, dtype=tf.float32), (2, 3))
output = input1 + input2

with tf.Session():
  result = output.eval()
result
```

```
array([[2., 3., 4.],
       [5., 6., 7.]], dtype=float32)
```

Export to other formats: Jupyter Notebook documents are easily accessed via a browser. However, it is s metimes useful or even necessary to have its contents in a different format, as a Python file for instance. In order to ease this conversion, Jupyter includes a tool named nbconvert. It is possible to convert a notebook to different static formats, that is, formats where the cells cannot be executed. These include: HTML, LaTeX, Markdown, reStructuredText, executable Python scripts. It is also possible to convert to a presentation format, which requires the previous installation of Pandoc via a package manager or their webpage.

Probably the main added value of Jupyter is the ability to create notebooks to provide conversations with data or computational narratives. Jupyter allows us, as educators, to narrate a "conversation between the student and data". The opportunity of intermingling computation into a narrative, creating a conversation with data is a powerful and effective form of communication. With Jupyter, you now have a new form of content to create and share with learners: ***computable content***. In a world where every subject matter can have a data-supported treatment, where computational devices are omnipresent and pervasive, the union of natural language and computation creates compelling communication and learning opportunities. Hereafter there is a list of the most common ways a Jupyter notebook can be used by the teacher:

Jupyter notebooks as textbooks. Instructors often write Jupyter notebooks as linear narrative documents. These notebooks are to be read by students and learners, perhaps worked through, marked up and are a relatively one sided information consumption experience

Notebooks as workbooks/primer. Workbooks engage students in the notebook environment by including active elements where they are asked to manipulate or create new content. These workbooks can be assigned as independent student learning (for example, pre-work for flipped classroom), or as part of an in class activity for individuals or small groups. A variety of activities can be supported with the executable code cells so learners can explore the space in an interactive and iterative environment. They can see and inspect portions of the surrounding code, but aren't required to touch it, maintaining appropriate granularity for assignments and challenges.

Notebooks as worksheets/drill sets. The cell based nature of the Jupyter notebook makes interactive code-based worksheets a clean experience for students to run. Each problem prompt can be written in a markdown cell, perhaps referencing an object or data file established in an initial cell. For example, a list or other data structure would be defined at the top and the exercises below are focused on the relevant methods and usage syntax. Each answer would then be completed by the student in a single code cell just below that markdown cell. This means that outputs and errors stay with the code producing them, so successes or bugs are easily traceable to the source. Usages of autograding tools or unit tests can be added to give students instant feedback about their work. Example or desired outputs could also be reproduced in markdown cells with the question for further guidance.

Notebooks as notepaper or course packets. The ability for a notebook to represent a linear experience with human prose and working code means that these can be used a student's notepaper in class. They can capture the linear narrative structure of a lesson or lecture, and actually run the code they are taking note of. This ensures that what they have written down actually works, and makes for a strongly reusable document for them moving forward with homework. Encouraging students to use Jupyter notebooks for taking class notes opens up further opportunities to provide scaffolding and support within the classroom.

Notebooks as an app. Notebooks even have a place in non-coding classroom content or activity. Interactive user inputs like mouse or touchscreen controlled sliders, buttons, highlighting, etc. allow a notebook user to manipulate input parameters for a visualization, tool, or model without directly editing any of the code within the notebook. These strategies support interactive computational exploration, or transform the notebook into an advanced calculator tool for students to use within their homework. These notebooks are then treated as applications that are distributed or made available for students to use during class or explore on their own.

Notebooks as lab reports or assignments. There are a variety of assignment deliverables that programming and technical courses may require. Students may be asked to produce essays, presentations, working code, analytics, and even art or music. Many of these deliverables are directly supported within the notebook environment. Any written work could be completed within the notebook environment with markdown, which is ideal for communication content that is driven by data or incorporating code content. For example, a student could write a computational essay within a notebook, and use one of the presentation tools to present a report out in class, all using the same notebook.

Notebooks as interactive multimedia platforms. A variety of media formats can be embedded within a notebook, and other tools more offer platforms to more directly connect notebooks with multimedia content. Instruction content might be split up between short videos (often for flipped classrooms) or a variety of static images might be important for an assignment. The markdown cells within the Jupyter notebook provide several ways to place hyperlinks and embed a variety of media. Several widgets are also available for embedding playable audio and video content (including from streaming video services) directly within the notebook. This creates a cohesive platform experience for the student, so they don't have to exit out or change screens to work on their assignment and reference that content.

Notebooks as a demonstration platform. This may be as a demonstration of how to use a notebook, presenting more a traditional style lecture, creating and editing code, or using an interactive feature to explore an experiment. Normal standards for font sizes, organization, and accessibility stand for these cases. This content may include text from markdown and LaTeX, code, and independent figures and sketches. Custom styling plugins are available to change the background colour, font, and other viewing aspects of the notebook for better presentation quality and accessibility. Several slide show tools are available, which allow you to markup notebook cell content for a more traditional slideshow presentation mode without having to exit from your standard notebook. An alternate modality such as a physical or digital "board" for free-form diagramming, working through a mathematical derivation, or other written procedural tasks can be useful. Notebooks can be a portion of these presentations or the complete environment, depending on your personal instruction style and content needs.

Notebooks as a live coding environment. Live coding involves the active writing of code within the instruction process. Introduction of bugs (either purposeful or accidental) to the code has the added benefit of giving the presenter an opportunity to work through the debugging process and demonstrating that perfect code is never created on the first go.

Live-coding can also be an opportunity to provide an active learning experience by providing notebooks with code that has not been completed before the lecture and having students attempt to fill in the missing lines before doing the live-coding demonstration. Feedback on where students are in this process can be a useful way to also judge what students are retaining and are struggling with leading to just-in-time teaching opportunities.

Formative assessment and prediction prompts can also be incorporated either directly into the notebook or as part of the narration of the lecture. Presentation styles of scrolling or shift + enter are not live coding, but live demonstrations. While these limit or negate the benefits of the live coding environment, the benefits of speeding up the presentation or running through code that's irrelevant to the learning goals may be more important.

Here after a collection of cells patterns that are particularly aligned with teaching and learning with Jupyter is introduced.

Shift-Enter for the win: Learners read and execute code, as well as potentially interact with a widget to explore concepts. Starting from a complete notebook, the instructor or learner runs through the notebook cell-by-cell by typing SHIFT + ENTER.

Fill in the blanks: To focus attention on one aspect of a workflow, the scaffolding and majority of the workflow can be laid out and some elements removed with the intent that students (or the instructor during a demo) fill in those pieces. The exercise might be accompanied by a small test that the code should pass, or a plot, or value which the code should generate if correct.

Target Practice: Focuses the learner's attention on one component of a multi-step workflow. The instructor provides all workflow steps except the one which is the focus of the exercise; the student will implement the "target" step within a notebook.

Tweak, twiddle, and frob: Students are given a notebook with a working example. They start by reading the text, running the code, and interpreting the results. Then they are asked to make a series of changes and run the code again; the changes can be small (tweaks), medium-sized (twiddles), or more substantial (frobs). Offering manipulations on a range of scales allows students to interact with notebooks in ways that suit their background and styles.

Win-day-one: A win-day-one exercise brings learners to the answer quickly and concisely, almost like a magic trick, and then breaks down and methodically works through each of the steps, revealing the magician's tricks. It generally involves multiple notebooks: the first notebook being the "win" which shows the workflow end-to-end, and subsequent notebooks breaking down the details of each component of the workflow.

Top-down sequence: Particularly in STEM, the default sequence of presentation is bottom-up, meaning that we teach students how things work (and sometimes prove that they work), before students learn how to use them, or what they are for. Notebooks afford the opportunity to present topics top-down; that is, students learn what a tool is for and how to use it, before they learn how it works.

Two bites at every apple: This pattern involves writing an activity that can address multiple audiences from different perspectives at the same time. This can be powerful when addressing a mixed audience of students.

Co-funded by the Erasmus+ Programme of the European Union

Coding as translation: Converting mathematics to code is a critical skill today that many students, especially those without strong programming backgrounds, struggle to do. Explicitly taking an equation and translating it step-by-step to the code can help these students make the transition to attaining this skill.

Symbolic math over pencil + paper: The objective is to convey an understanding of a physical system governed by a complicated mathematical system. Working out the algebra is necessary to uncover the fundamental behaviour of the system, but how to do the algebra is not the goal of the lesson. In this case, you want to see the algebraic result and then teach the students the underlying meaning of the system.

Replace analysis with numerical methods: Some ideas that are hard to understand with mathematical analysis are easy to understand with computer simulation and numerical methods.

The API is the lesson: When students work with a software library, they are exposed to functions and objects that make up an application programming interface (API). Learning an API can be cognitive overhead; that is, material students have to learn to get work done computationally, but which does not contribute to their understanding of the subject matter. But the API can also be the lesson; that is, by learning the API, students are implicitly learning the intended content.

Proof by example, disproof by counterexample: In many classes, students see general results derived or proved, and then use those results in programs. Notebooks can help students understand how these results work in practice, when they apply, and how they fail when they do not.

The world is your dataset: A hazard of this pattern is that students can spend too much time looking for data that is not available. They might need coaching about how to make do with the data they can get, even if it is not ideal.

Now you try (with different data or process): Students start with a complete working example provided by an instructor and then they change the dataset or process to apply the notebook to an area of their own choosing. This method can allow more or less fluctuation depending on the skills of the students.

Connect to external audiences: This is in some sense the opposite of “the world is your dataset.” Here the goal is to take a workflow or computational exploration and share it with the world so others can see it, learn from it, reuse and remix it.

There can be only one: This pattern involves creating a competition between individual students or teams of students. Clear goals and metrics need to be defined and then students submit notebooks that are scored and evaluated. Competitions can span months or be completed in a single class.

Hello, world!: In some situations (such as the first day of class of a very introductory course) you may wish to do no more (and no less) than build confidence in the students’ abilities to be able to write a first computer program. Traditionally, the first program written was a “hello, world” program: a program that did nothing but display the text “hello, world” on the screen.

Test driven development: The instructor provides tests written in a unit testing framework like unites or doctest; students write code to make the tests pass. This pattern requires the overhead of teaching students about the unit testing framework. Students working to make tests pass can lose their view of the big picture, and feel like they have been robbed of autonomy. This type of exercise is best used sparingly.

acadigia
Puts the digital in academia
Co-funded by the Erasmus+ Programme of the European Union

Code reviews: Code reviews involve a student or instructor providing feedback on someone else's code. This pattern involves peer work as well as a means for providing feedback to students on topics other than correctness of their code but also on code readability and styling.

Bug hunt: The instructor provides a notebook with code that contains deliberate bugs. The students are asked to find and fix the bugs. Automated tests might be provided to help students know whether some bugs remain unfixed.

Adversarial programming: This pattern involves participants writing a solution to a problem and tests that attempt to make the written solution fail. This pattern can be done in many ways including having students complete the tasks and pair up and exchange solutions/tests or having the instructor writing the solution and the students then write the tests.

"ticket to leave" One example of generating participation in the classroom with Jupyter notebooks is the Activity magic, available as an extension. It creates what has been called a "ticket to leave" (or "exit ticket") via the notebook. The idea of a "ticket to leave" is an excellent way to end a class or lab. Briefly, it is just a survey that you give the students (see figure). Often, these surveys are given via a Personal Response System (also known as "clickers" or PRS) or cell phones. These questions do not typically require much time to answer, but are meant to capture the essence of the conversation of the class. After a minute or so to contemplate the question, the students select their answer (by clicking one of the buttons), and instructor shows the gestalt results.

3.4 Supporting Assessment

Professional handle and grade of the assessment of a notebook-based submission is achieved with Nbgrader, a tool that facilitates creating and grading assignments in the Jupyter notebook. It allows instructors to easily create notebook-based assignments that include both coding exercises and written free-responses. Nbgrader then also provides a streamlined interface for quickly grading completed assignments.

Nbgrader includes **two tools**. On the one hand, an additional toolbar to each cell in Jupyter Notebook to choose if that cell corresponds to the **instructions of the assignment**, if it will include the student's answer, or if it will be a marking cell with the possibility to include code to mark automatically those exercises. On the other hand, it generates a new tab denominated **Formgrader** in the Jupyter Dashboard. For the teacher, it allows to **assign** tasks, **validate** those tests that have undergone automatic marking, **retrieve** exercises sent by the students, **mark** them and **supply** the marked versions **back** to the students.

Main functionalities supporting assessment are described hereafter.

- **Auto grading**: Nbgrader allows code cells in a notebook to be marked to be auto-graded or manually graded. An instructor can then create an assignment that can be completely auto-graded, requiring little work after the notebook has been created. This makes grading much easier and scales well with large class sizes.

  Autograding is very simple in principle, it only requires to run the notebook. The actual effect is no different than the "Restart and run all cells" functionality within the Jupyter interface. The difference is that, after running, it looks for cells that have an error output. If any of these cells are marked as "autograder tests", then these cells have a point value, and that point value is subtracted. Error output is simple any text on the standard error stream, which is saved separately within the notebook output from the standard output stream. It is up to the Jupyter kernel to write an error message to the standard error stream, otherwise autograder doesn't work (this has been a problem with a few languages kernels in the past).

The teacher can also create manually graded cells for a portion of an assignment and provide written feedback to the student. This allow the inclusion of questions that cannot be auto-graded, such as reflection questions.

- **Manual grading**: After auto grading, there is a web UI (via the formgrader extension) to do manual grading. This allows one to see the output from autograding, give comments, adjust points, etc. There are also purely manually graded exercises. The output from manual grading is only stored in gradebook.db, and is merged into the final output at the feedback step.

- **Validation**: Another assessment function very related to autograding is "validation". There is a button on the student interface marked "validate", which executes the student version of the notebook from top to bottom, and reports any errors. This is exactly equivalent to "Restart and run all", but doesn't stop on errors. Since all it can access is the actual notebook file the student has, it can not take into account the hidden tests. If an instructor wants a test to be visible to the students.

- **Student grades management** : It is performed through the gradebook or grades database. The gradebook or database is stored (by default) at gradebook.db at the root of the course directory. gradebook.db stores students and grades. Less obviously, it stores the assignments and the contents of hidden or immutable cells, such as the contents of the hidden tests or read-only cells. This is used to restore these cells when students return them.

  First, the gradebook stores student mappings. It stores a student_id (string) that is the name used on the filesystem for each student. It can also store a firstname/lastname/email for each student, but it doesn't try to replace a complex student management system. The database also stores assignments and their cells. For example, it stores the contents of read-only cells, and autograder tests cells, which get re-inserted into the notebook before the autograde step. Cells are stored by the cell ID, which is in the cell metadata (cell metadata is a ipynb-format native concept). The autograder step looks at the database and re-inserts data based on the cell ID. In the formgrader "manual grading" interface, the instructor can manually grade assignments (after autograding), and these points + comments are added to the database.

- **Export grades:** Grades can be exported in csv format. You can also build other exporters, which access the database and export somehow - to a file, or perhaps other fancy things like uploading directly.

There are several interfaces, or web extensions, that can be directly used directly from Jupyter as the "default" ways of using nbgrader.

- The **Assignment list extension** serves as the student-facing interface for the notebook file browser view. It fetches assignments from the exchange directory, allows students to open them, and submit them back to the exchange. This is for the Jupyter notebook file-browser view
- The **formgrader extension** is the instructor-facing interface accessible from the file browser view. It allows the instructor to browse assignments, open them, manage students, etc. This is for the Jupyter notebook file-browser view.
- The **validate extension** is a student-facing for the notebook view that does validation. Basically, it is the same as "Restart and run all cells" but it shows errors a little bit nicer.

- The **create assignment extension** is an instructor-facing for the notebook view. It provides a toolbar that allows you to edit cell metadata.

Nbgrader has also some commands that make possible to manage assignments and grades using command code lines from the console, such as Nbgrader generate_assignment (converts the release notebook file from the source), Nbgrader autograde (to re-insert the read-only cells , replace them with the known-good versions, and execute the entire notebook), Nbgrader generate_feedback (take any feedback created during manual grading, for all student submissions in this assignments, and create a .html file), etc,..

3.5. Supporting the Empowerment of Learners

The following characteristics support the empowerment of the learners.

- Jupyter Notebook is a great tool for **designing interactive seminars**, setting them as a new teaching tool that can be access in class and through the Virtual Campus. The European Space for Higher Education (ESHE) Committees have elaborated protocols where great emphasis is put on innovation towards improving classroom teaching (theory and practice) with assignments and attendance to seminars. In particular, interactive seminars allow students to acquire a theoretical knowledge along with a practical one.

- Jupyter platform for symbolic and numerical calculations allows for a natural coexistence between explanatory text, command lines and plotting. This in turn helps lets students to **use this environment without needing to understand all of its technicalities.** In the Sciences, it is also crucial to provide students with programming resources in order to endow them with the numerical tools needed for high-complexity scientific problems. This way, time is spent more in the conceptual ideas, rather than in tedious calculations.

- **Different levels of complexity can be considered depending on the teaching needs and the student ́s motivation.** The first level would consist on using explanations supported by plots obtained from simulations. The next level would allow students to play with the simulations as if these where black boxes, that is, by tuning different parameters and obtaining results without getting into the computational methods. A more advanced level would allow students to modify code to increase the given simulations, letting them acquire new computational abilities.

- Jupyter Notebook **is not circumscribed to teaching purposes**. Their modules and functions can also be applied in a more advanced research environment, substituting partially or completely the use of other comercial software for symbolic and numerical calculations. Thus, learning how to use Jupyter Notebook can also be beneficial in the future career of students.

- **Improve effective communication skills.** Notebooks can also help teach effective communication skills, combining prose with graphics into a strong narrative.

- **Self-learning is encouraged** to different levels of complexity depending on the student ́s needs and interests.

- Thanks to possibility to embed of images, HQ video and links to websites, **learning is much more appealing.**

- **The Virtual Campus**, a platform that more and more students are getting used to, **is enriched with new possibilities.**

Co-funded by the
Erasmus+ Programme
of the European Union

- **Marking and automatic marking can be done remotely in a more flexible manner**, tailored to the particular needs of each student. This frees classroom time for the teacher to delve more in depth into the explanations.

- Once students have the software on their computers, they always have access to it; they can work anywhere, and they can use it for internships, jobs, and other non-school activities. It is easy for them to install additional packages later.

3.6. Facilitating Learners' Digital Competence

- **Increase computational thinking:** Jupyter notebooks support a wide range of learning goals. Its interactivity enables building intuitive understanding of domain knowledge, such as the understanding of a mechanical response of a system while varying parameters or understanding how an algorithm behaves. Using notebooks, you can create rich learning experiences that link together the core foundations of computational thinking:
  - Decomposition: Breaking down data, processes, or problems into smaller, manageable parts
  - Pattern Recognition: Observing patterns, trends, and regularities in data
  - Abstraction: Identifying the general principles that generate these patterns
  - Algorithm Design: Developing the step by step instructions for solving this and similar problems

- **Programming skills.** Notebooks can support teaching or strengthening programming skills, by combining code with text descriptions and visualizations. Even if a notebook is designed to be consumed passively, the exposure to code helps show students how to do something—and that they can do it themselves. This also helps demystify coding for students who do not view themselves as traditional "computer science" types.

- **Learning new open source programming languages is a natural outcome**. This in turn allows to learn software that is free of charge, flexible and easily shareable and exportable. The Jupyter system supports over 100 programming languages (called "kernels" in the Jupyter ecosystem) including Python, Java, R, Julia, Matlab, Octave, Scheme, Processing, Scala, and many more. Jupyter's kernel flexibility allows instructors to pick the right language for a particular context. For example instructors may use Python to teach programming, while switching to R to teach statistics, and then perhaps Scala to teach big-data processing

- **Kernel independent interface**. Regardless of the language chosen, the Jupyter interface remains the same. Thus, some cognitive load can be lessened when using multiple languages within or across courses (e.g., the user interface stays the same between the student's Digital Humanities and Biology courses). Students often appreciate consistent use of the same language within a course, however.
- **Learning LaTeX is fostered no end**, a very beneficial outcome due to the massive use of this format for elaborating scientific documents or even for printed material.

- **Active acquisition of numerical calculation abilities is greatly enhanced** Jupyter Notebook as an educational tool is particularly relevant to those studies with a scientific or technical component, where numerical, symbolic and statistical calculations are routine. Nonetheless, it can also be relevant to other university studies.

- **Exposure to Open-source tool.** Integrating notebooks into classes also exposes students to a large and growing ecosystem of open-source tools. This supports their education, but also provides experience in the same environment of tools used in industries in high demand for trained employees, such as data science and machine learning. The open-source nature of these tools also ensures that course content remains accessible and affordable to all students—including those outside the traditional university environment.

- Students **learn to install and set up Jupyter**, and software in general, which is a skill they are likely to need. Students learn to use Jupyter on their preferred OS, e.g. Linux, Mac, or Windows, which means they are already familiar with the basic idioms of their OS.

- The **total computing power for the class scales** with the number of students, as long as each student has enough CPU power and memory to support the intended applications.

Summary of functionalities	
1. Notebook as aa new form of data-based communication	A notebook is a document that supports mixing executable code, equations, visualizations, and narrative text. Because of the mix of code and text elements, these documents are the ideal place to bring together an analysis description, and its results, as well as, they can be executed perform the data analysis in real time. Main uses of notebooks include: textbooks, workbooks/primer, worksheets/drill sets, notepaper or course packets, an app, lab reports or assignments, interactive multimedia platforms, demonstration platform, live coding environment, etc,..
2. Creation of computational stories	Jupyter notebooks allows the user to bring together data, code, and prose, to tell an interactive, computational story. the notebooks can combine explanations traditionally found in textbooks with *the interactivity of an application*.
3. Web application	The web application is a browser-based tool for interactive authoring of documents which combine explanatory text, mathematics, computations and their rich media output. As a “web application”, you can create and share documents that contain live code, equations, visualizations as well as text. As a server-client application, the Jupyter Notebook App allows you to edit and run your notebooks via a web browser
4. kernel	A kernel is a program that runs and introspects the user’s code. The Jupyter Notebook App has, by default, a kernel for Python code, but there are more than a 100 kernels available for other programming languages.
5. Dashboard	The dashboard of the application not only shows you the notebook documents that you have made and can reopen but can also be used to manage the kernels: you can which ones are running and shut them down if necessary.
6. Usage by a large number of students	This educational projects are available without needing to install commercial software
7. Accessibility	It can be accessed through different devices, such as smartphones or tablets
8. Installation	Jupyter permits local installation on students’, on lab computers Jupyter or on remote servers

Co-funded by the Erasmus+ Programme of the European Union

9. **Operating systems**	Jupyter is cross-platform and ideally behaves the same on Windows, Mac, or Linux, and distributions such as Anaconda
10. **Distribution and collection of materials**	Jupyter notebooks are plain text computer files, so you can distribute them to students and collect them using any system that handles text files, including GitHub, Google Drive, and (as a last resort) email attachment
11. **Usability on Learning management systems**	Most LMS tools are not yet notebook-aware, so they don't render notebooks or make it easy for instructors to comment on or grade them. Some tools and workflows are being actively developed to connect the Jupyter ecosystem to the LMS ecosystem using the Learning Tools Interoperability (LTI) standard
12. **Web hosting**	Notebooks can be publicly hosted on any website, so students can download the files by clicking on a link. Most web-hosting software is not notebook-aware, but you can use nbviewer to share public notebooks, rendered as a static web page. nbviewer is a web service provided by Project Jupyter.
13. **Compatibility with GitHub**	One of the popular tools for distributing and collecting notebooks is GitHub, a hosting and collaboration platform for software. GitHub is based on git, a version-control system. Educators at academic institutions can use GitHub Classroom, which allows instructors to set up assignments for a class.
14. **JupyterHub**	Allows the teacher to place notebooks and any related files directly into the students' directories manually or via a script
15. **Materials management**	Jupyter notebooks can be used to organize classroom materials and objects, store and provide access to reading materials for students, present and share lecture materials, perform live coding, explore and interact with materials, support self-paced learning, grade students' homework, solve homework problems, or make materials reusable to others.
16. ***Cells***	The entire contents of a notebook is composed of only cells. These cells can take one of two forms: text or code. Code cells are composed of three areas: the input area, the display area, and the output area. Cells patterns particularly aligned with teaching and learning include: "Shift-Enter for the win"; "Fill in the blanks"; "Target Practice"; "Tweak, twiddle, and frob"; "Win-day-one"; "Top-down sequence"; "Two bites at every apple"; "Coding as translation"; "Symbolic math over pencil + paper"; "Replace analysis with numerical methods"; "The API is the lesson"; "Proof by example"; "The world is your dataset"; "Now you try"; "Connect to external audiences"; "Here can be only one"; "Hello, world"; "Test driven development"; "Code reviews"; "Bug hunt"; "Adversarial programming"; "Ticket to leave"
17. **Widgets**	Widgets are "mini" Graphical User Interfaces (GUI) that give the notebook user access to slide bars, toggle buttons, and text-boxes
18. **Accessing documentation in the notebook**	From a notebook cell, the TAB key autocompletes and SHIFT-TAB brings up full documentation
19. **Magics**	Magics are meta-commands that only function within Jupyter and allow a user to access language/kernel-specific features.

20. Notebooks under version control	Keeping notebooks under version control is a great way to not only keep track of changes to your content, but also for sharing it
21. Testing notebooks	: Before distributing notebooks, at a minimum, you can test that the notebook executes cleanly from top to bottom by restarting the kernel and running all cells from top to bottom.
22. Advanced topic: extensions	There are many communities contributed extensions that add functionality to Jupyter notebooks. Extensions vary from displaying an automated table of contents for a notebook, or prettify code, or hiding/showing solution cells.
23. Export to other formats	It is possible to convert a notebook to different static formats, that is, formats where the cells cannot be executed. These include: HTML, LaTeX, Markdown, reStructuredText, executable Python scripts. It is also possible to convert to a presentation format, which requires the previous installation of Pandoc via a package manager or their webpage
24. Formgrader	It allows to set an manage the instructions of an assignment, e.g. if it will include the student's answer, or if it will be a marking cell with the possibility to include code to mark automatically those exercises.
25. Formgrader Dashboard	It allows to assign tasks, validate those tests that have undergone automatic marking, retrieve exercises sent by the students, mark them and supply the marked versions back to the students.
26. Auto grading:	Nbgrader allows code cells in a notebook to be marked to be auto-graded or manually graded. An instructor can then create an assignment that can be completely auto-graded, requiring little work after the notebook has been created
27. Manual grading	After auto grading, there is a web UI (via the formgrader extension) to do manual grading. This allows one to see the output from autograding, give comments, adjust points, etc. There are also purely manually graded exercises.
28. Validation	There is a button on the student interface marked "validate", which executes the student version of the notebook from top to bottom, and reports any errors.
29. Marking and automatic marking	Marking and automatic marking can be done remotely in a more flexible manner, tailored to the particular needs of each student. This frees classroom time for the teacher to delve more in depth into the explanations.
30. Student grades management	It is performed through the gradebook or grades database. The gradebook or database is stored (by default) at gradebook.db at the root of the course directory. gradebook.db stores students and grades. It stores the assignments and the contents of hidden or immutable cells, such as the contents of the hidden tests or read-only cells. This is used to restore these cells when students return them.
31. Export grades	Grades can be exported in csv format. You can also build other exporters, which access the database and export somehow - to a file, or perhaps other fancy things like uploading directly.
32. Assignment list extension	The Assignment list extension serves as the student-facing interface for the notebook file browser view. It fetches assignments from the exchange directory, allows students to open them, and submit them back to the exchange. This is for the Jupyter notebook file-browser view.

Co-funded by the Erasmus+ Programme of the European Union

33. formgrader extension	The formgrader extension is the instructor-facing interface accessible from the file browser view. It allows the instructor to browse assignments, open them, manage students, etc. This is for the Jupyter notebook file-browser view.
34. validate extension	The validate extension is a student-facing for the notebook view that does validation. Basically, it is the same as “Restart and run all cells” but it shows errors a little bit nicer.
35. create assignment extension	The create assignment extension is an instructor-facing for the notebook view. It provides a toolbar that allows you to edit cell metadata.
36. symbolic and numerical calculations	Jupyter platform for symbolic and numerical calculations allows for a natural coexistence between explanatory text, command lines and plotting. This in turn helps lets students to use this environment without needing to understand all of its technicalities.
37. levels of complexity	Different levels of complexity can be considered depending on the teaching needs and the student΄s motivation.
38. Not circumscribed to teaching purposes	. Their modules and functions can also be applied in a more advanced research environment, substituting partially or completely the use of other comercial software for symbolic and numerical calculations. Thus, learning how to use Jupyter Notebook can also be beneficial in the future career of students.
39. Multimedia resources	It is possibility to embed of images, HQ video and links to websites, making learning is much more appealing
40. Increase computational thinking:	Using notebooks, you can create rich learning experiences that link together the core foundations of computational thinking: Decomposition, Pattern Recognition, Abstraction and Algorithm Design
41. Programming skills	Notebooks can support teaching or strengthening programming skills, by combining code with text descriptions and visualizations. Even if a notebook is designed to be consumed passively, the exposure to code helps show students how to do something—and that they can do it themselves. This also helps demystify coding for students who do not view themselves as traditional “computer science” types.
42. Learning new open source programming languages is a natural outcome	The Jupyter system supports over 100 programming languages (called “kernels” in the Jupyter ecosystem) including Python, Java, R, Julia, Matlab, Octave, Scheme, Processing, Scala, and many more. Jupyter’s kernel flexibility allows instructors to pick the right language for a particular context.
43. Kernel independent interface	Regardless of the language chosen, the Jupyter interface remains the same. Thus, some cognitive load can be lessened when using multiple languages within or across courses (e.g., the user interface stays the same between the student’s Digital Humanities and Biology courses). Students often appreciate consistent use of the same language within a course, however.
44. Latex	Learning LaTeX is fostered no end, a very beneficial outcome due to the massive use of this format for elaborating scientific documents or even for printed material.
45. Active acquisition of numerical calculation abilities	Jupyter Notebook as an educational tool is particularly relevant to those studies with a scientific or technical component, where numerical, symbolic and statistical calculations are routine.

46. Exposure to Open-source tools	Integrating notebooks into classes also exposes students to a large and growing ecosystem of open-source tools. Students learn to install and set up Jupyter, and software in general, which is a skill they are likely to need. Students learn to use Jupyter on their preferred OS, e.g. Linux, Mac, or Windows, which means they are already familiar with the basic idioms of their OS.
47. Computing power	The total computing power for the class scales with the number of students, as long as each student has enough CPU power and memory to support the intended applications.

4. Tutorial Video and PowerPoint slides

The ACADIGIA resources for Blackboard are available on the website here.

Co-funded by the Erasmus+ Programme of the European Union