

Community teaching training

📌 Under development (mid 2022)

As of mid-2022, this material has been reworked for our summer workshop and for the CarpentryCon 2022 workshop. The material will continue to be improved before a longer training session in the autumn.

This material is the new version of our previous [Instructor training](#).

What this course covers

In 2020, we got farther from our offices but closer to each other. If you could adapt to the circumstances, a huge number of possibilities were opened - in particular with collaborative teaching. In this training, you will learn how we use them. Teaching from 2019 is almost obsolete - join us to learn more.

- **Tools of teaching:** how to make the most out of online (and other) teaching.
- **Workshop organization, collaboratively:** our vision of teaching together outside of our silos.
- **Socio-technical factors:** social and technical barriers to learning, why you need to care, and what you can do about them.
- **Lesson development, collaboratively:** how to design lessons and teaching materials so that they can be open and shared.

Who is the course for?

Teaching is a profession, but also something that everyone needs to be able to do to some degree, since everyone has their own personal specialties they will either teach or mentor.

This course can be relevant for different learner personas:

- **You run a practical teaching program** at your institution (for example as part of a research computing group) and would like to learn best practices for collaborative teaching, so that you aren't re-inventing the same thing over and over again.
- **You are a technical specialist** who is frustrated with the way you currently try to teach others who need to use your software or infrastructure. You can't spend too much time to become a professional, but you know you need something more than what you've been

doing. Thus, you would like to adopt some of the best practices of designing and teaching interactive, hands-on workshops.

- **You've been teaching alone**, but would like to join a collaboration network for more co-teaching and to reduce the amount of duplication of effort.
- **You are interested in teaching CodeRefinery lessons**, and would like a comprehensive kick-start to how CodeRefinery works, either to join us, or teach its lessons with us or independently.

Pre-workshop preparation

These things will help you get the most out of this workshop, by giving you a broad overview at the beginning that you might only get later on in the lesson.

Don't worry if you don't have time to do everything here. The most important ones are listed first.

Read "How to help someone use a computer" (5 min)

[How to help someone use a computer](#), by Phil Agre. Summary: Most of our teaching challenge is helping people to overcome bad user interface design.

Browse a CodeRefinery lesson (5 min)

Please take 5 minutes and go through a [CodeRefinery lesson](#) and understand the general layout. Don't go in-depth to any of the material (unless you want, obviously). We would recommend [git-intro](#) if you don't have a preference.

(optional) Watch "The future of teaching" (35 min content only, 45 min with Q&A, or 15 min reading)

The "The Future of Teaching" talk is a summary of many things in this course. You could watch it as a preparation (or if you are reading on your own, as a summary). This is low priority, since we will cover most of these things in the course itself. [Watch it on YouTube](#) or [read it](#) if you prefer.

(optional) Read "The science of learning" (20 min)

Read this short paper [The Science of Learning](#) which provides a brief overview of some key evidence-based results in teaching. This paper is also used by [The Carpentries](#) for their Instructor Training workshops.

Welcome and introduction

 What do we want to get out of this workshop

- Introduction of instructors and helpers
- Each instructor can say what we want to get out of the instructor training
- But we want to know from everybody and collect these in the live notes

Goals for this workshop

Goals for this instructor training

- **Inspire teachers and staff who have to teach indirectly as part of their job:** use best practices for the modern world, especially for online teaching.
- **Promote collaboration in teaching:** less going alone.
- **Promote CodeRefinery sustainability:** form a network that can work together to share the work and benefits.

Giving confidence

Goal number one should be that we give participants the confidence to independently apply the tools or knowledge learnt. This is more important than giving a “complete” overview. [Lucy Whalley gave this great comment at one of our workshops]

- You don't have to know everything to use (or teach) something.
- For the large majority of topics we teach, there are many resources online which provide how-to guides or tutorials. And the Stack Overflow answer bank isn't getting any smaller. So we need to ask why do people attend in-person sessions if there is information freely available? Our impression is that it is for confidence building, identity formation, perhaps signposting to resources.
- This also links with building a welcoming/inclusive environment: for example, imposter syndrome, which disproportionately affects under-represented groups ([link](#)), can manifest as low self-confidence → building the confidence of students in the classroom may lead to a more diverse community.

Tools for this workshop

We often start workshops with these:

- [HackMD mechanics and controls](#)
- [Zoom mechanics and controls](#) (but this got less relevant since starting teaching via live-streaming where participants only interact via HackMD)

Code of Conduct

- We follow [The CodeRefinery Code of Conduct](#).
- This is a hands-on, interactive workshop.
 - Be kind to each other and help each other as best you can.
 - If you can't help someone or there is some problem, let someone know.
 - If you notice something that prevents you from learning as well as you can, let us know and don't suffer silently.
- It's also about the little things:
 - volume
 - font size
 - generally confusing instructor
 - **not enough breaks**

About the CodeRefinery project and CodeRefinery workshops

Keypoints

- Teaches intermediate-level software development tool lessons
- Training network for other lessons, too
- Publicly-funded discrete projects (3 projects actually) transitioning towards an open community project
- We have online material, teaching, and exercise sessions
- We want more people to work with us, and to work with more people

CodeRefinery is a [Nordic e-Infrastructure Collaboration \(NeIC\)](#) project that has started in October 2016 and is funded until February 2025.

The funding from 2022-2025 is designed to keep this project active beyond 2025 by forming a support network and building a community of instructors and contributors.

History

The CodeRefinery project idea grew out of two [SeSE](#) courses given at KTH Stockholm in 2014 and 2016:

- <http://sese.nu/scientific-software-development-toolbox/>
- <http://sese.nu/scientific-software-development-toolbox-2016/>

The project proposal was submitted to NeIC in 2015, accepted in 2015, and started in 2016.

We have started by porting own lessons to the Carpentries teaching style and format, and collaboratively and iteratively grew and improved the material to its present form.

Main goals

- Develop and maintain **training material on software best practices** for researchers that already write code. Our material addresses all academic disciplines and tries to be as programming language-independent as possible.
- Provide a [code repository hosting service](#) that is open and free for all researchers based in universities and research institutes from Nordic countries.
- Provide **training opportunities** in the Nordics using Carpentries and CodeRefinery training materials.
- Articulate and implement the CodeRefinery **sustainability plan**.

Impact

We collect feedback and survey results to measure our impact.

3-6 months after attending a workshop, past participants are asked to complete a short post-workshop survey. The survey questions aim to establish what impact CodeRefinery workshops have on how past participants develop research software.

Pre- and post-workshop survey results

- Overall quality of research software has improved: more reusable, modular, reproducible and documented.
- Collaboration on research software development has become easier
- Past participants share their new knowledge with colleagues
- Usage of several tools is improved, and new tools are adopted

[Free-form answers](#) also suggest that workshops are having the intended effects on how people develop code. A common theme is:

I wish I had known this stuff already as a grad student 10+ years ago...

We would love to get suggestions on how we can better quantify our impact. This would make it easier for us to convince institutions to partner with us and also open up funding opportunities.

Target audience

Carpentries audience

The Carpentries aims to teach computational **competence** to learners through an applied approach, avoiding the theoretical and general in favor of the practical and specific.

Learners do not need to have any prior experience in programming. One major goal of a Carpentry workshop is to raise awareness on the tools researchers can learn/use to speed up their research.

By showing learners how to solve specific problems with specific tools and providing hands-on practice, learners develop confidence for future learning.

Novices

We often qualify Carpentry learners as **novices**: they do not know what they need to learn yet. A typical example is the usage of version control: the Carpentry `git` lesson aims to give a very high level conceptual overview of Git but it does not explain how it can be used in research projects.

CodeRefinery audience

In that sense, CodeRefinery workshops differ from Carpentry workshops as we assume our audience already writes code and scripts and we aim at teaching them **best software practices**.

Our learners usually do not have a good overview of **best software practices** but are aware of the need to learn them. Very often, they know the tools (Git, Jupyter, etc.) we are teaching but have difficulties to make the best use of them in their software development workflow.

Whenever we can, we should direct learners that do not have sufficient coding experience to Carpentries workshops.

Competent practitioners

We often qualify CodeRefinery learners as **competent practitioners** because they already have an understanding of their needs. *Novices* and *competent practitioners* will be more clearly defined in the [next section](#).

Best software practices for whom?

It can be useful to ask the question: *best software practices for whom?* CodeRefinery teaches *best software practices* derived from producing and shipping software. These practices are also very good for sharing software, though our audience will probably not need to embrace *all* aspects of software engineering.

Teaching online

Is online teaching better or worse? As usual for that question, “it’s all a trade-off”. We believe that many people tried to directly translate in-person strategies to online, and teaching suffered. In some ways, this even revealed the flaws of in-person pedagogy (remove some interaction and not much is left other than an info-dump).



Exercise

Using HackMD, brainstorm advantages and disadvantages of online and in-person teaching. For each disadvantage of each, think of ways to compensate.

- We won't elaborate more here right now - most of the rest of the course somehow relates to online teaching!
- We will see many techniques which compensate for some of the disadvantages in the section **tools of teaching**.
- **Workshop organization** will discuss new collaborative opportunities with online teaching.

See also

- The rest of this course
- CodeRefinery manuals: <https://coderefinery.github.io/manuals/>

Team teaching

One of the most significant improvement of our teaching has been the concept of **co-teaching**.

! Co-teaching

Wikipedia: Co-teaching or team teaching is the division of labor between educators to plan, organize, instruct and make assessments on the same group of students, generally in the a common classroom,[1] and often with a strong focus on those teaching as a team complementing one another's particular skills or other strengths.

This is not strictly an effect of moving online. However, the larger number of instructors and larger audiences make this practical on a wide scale.

Primary article

<https://coderefinery.github.io/manuals/team-teaching/>

Benefits

- The course seems very interactive, much more so than expecting students to speak up. The co-teacher can take on the “voice of the audience”.
- Quicker preparation time since co-teachers can rely on each other in unexpected situations.
- One co-teacher can be effectively learning at the same time and thus acting as the “voice of the audience” in another way.
- Great way to onboard new instructors - extensive training and preparation no longer needed.
- More active minds means better able to watch and react to other feedback, such as HackMD or chat.

- Less workload - one person does not have to prepare perfectly, any uncertainty can usually be quickly answered by the other.

Strategies

In reality, these strategies are mixed and matched even within a lesson, and there are many things between these:

- One person gives lectures, one does the typing during demos.
- “Interview”: One primarily doing the “teaching”, one guiding by asking questions - either as an interviewer or as a virtual learner.

Things that don’t work (are not team teaching):

- Dividing up a lesson into parts, each person gives different parts independently.

Exercises

Exercise

Divide into groups of two or three. Choose one of the two models in the team-teaching page, and quickly (5 min) prepare a short topic (3-5 min) to team teach. You can quickly scan the “preparation” section at the bottom.

The challenge is not just to give the lesson, but to prepare the lesson quickly and rely on each other to give a good lesson anyway.

See also

(none yet)

HackMD

The name “HackMD” doesn’t do this concept justice, nor do the more common descriptions of “shared notes” or “collaborative document”. It is a full replacement for chat: perhaps it could be called **random access chat** or **parallel 2D chat** ?

HackMD itself is a web service for collaborative documents in Markdown. This isn’t special to HackMD, but we have used it for its scalability, markdown support, and privacy in the “view” mode.

Primary articles

- HackMD instructors for the audience: <https://coderefinery.github.io/manuals/hackmd-mechanics/>
- HackMD manager role description: <https://coderefinery.github.io/manuals/hackmd-helper/>

- Example published HackMD after six 3-hour CodeRefinery sessions:
<https://coderefinery.github.io/2022-03-22-workshop/questions/>

Advantages

- Synchronous questions, no disadvantage for quiet people.
- Anonymous questions.
- Parallel answers by a large number of helpers.
- Easier to go back and review past questions during Q&A sessions (compared to scrolling through chat), for example finding important or unanswered questions.
- The above can make a course feel much more interactive than it would otherwise.

Disadvantages

- *Overwhelming* flood of information
 - But you wanted more interaction, right?
 - Co-teaching helps here, one person can focus on watching.
 - Students must be warned to be deliberate about where they focus their attention (different learners have different interests).
- It is another tool to use
 - Not required for basic learners, learners can begin using when they are comfortable.

Exercise

Exercise

- Actively use HackMD during this course.
- Observe how the instructors integrate it during the course itself, and can immediately respond to the questions.
- Observe how instructors occasionally mention and screenshare HackMD to validate to the audience that it is being watched.
- Keep HackMD open. Can you balance it and watching. Does it increase or decrease engagement?

Teams

Everyone wants interaction in courses, yet when a group size gets too large, it doesn't have much interaction. A event in a physical space naturally makes teams based on who is nearby. When done online, this needs to be more explicit.

Primary articles

- Exercise leader role description: <https://coderefinery.github.io/manuals/exercise-leaders/>
- Local breakout rooms: <https://coderefinery.github.io/manuals/local-breakout-rooms/?highlight=teams>

Basic concepts

- Teams are pre-assigned
- **Exercise leaders** (aka helpers) assigned per team
- Teams stay together during the whole workshop.
- Learners can sign up either alone...
- ... or they can sign up with a **pre-made team**: people who know each. “bring your own breakout room”:
 - When two people in a work group learn a skill, uptake within the group is often much higher. Thus, we strongly encourage pre-made teams that know each other.
 - Teams that all come from the same group or field, with a helper from that field, can transition to help

Online

In the best online implementations, our teams have these properties:

- Coordination of breakout rooms is a *lot* of work.
- In zoom, we could request learners to rename to (N) Learner Name, and then quickly assign people. Now, you can have learners self-select their rooms. But will they actually do this, or stay in main room?
- One helper is assigned per team.
 - In fact, we would limit the number of registrations to 5× the number of helpers so that all teams have a helper
- Our registration system (indico) is capable of mailing personalized messages per person with their team information. This is quite a bit of work to manage.

But they have these disadvantages:

- Much, much harder registration coordination, almost to the point of being impossible.
- Number of attendees.
- Difficulties when attendees drop out partway through a course.

In-person

Teams may naturally form based on setting location, but

- Teams may happen naturally by sitting at the same table
- Do teams stay the same day after day?
- Do teams get arranged in a manner useful for learning?
- Do you have one helper per team?
- Do you encourage people to interact explicitly enough?
- Do you ensure that no one gets left out in the crowd? Are the teams explicit enough?

Discussion: what we actually do

- For large enough CodeRefinery workshops, assign teams with one helper each. Deal with re-adjustment

- The livestream option allows everyone else to follow along.
- In other workshops, create breakout rooms but somehow try let people self-assign. Most don't.
- For large workshops without enough staff help, livestream and encourage people to form their own teams and watch themselves - we don't actually need to be involved.
- Teams can be delegated to a local organizer.

Exercises

Teams

Consider these questions:

- Should teams have similar or different people in them?

Livestreaming

In a large lecture, in effect, you don't interact with most people - not even close. Yet, trying to force this limited interaction greatly limits our possible reach, and keeps us in an awkward state of expecting live interaction yet usually receiving minimal. When we try this online, it forces a closed

Teaching via livestreaming allows us to:

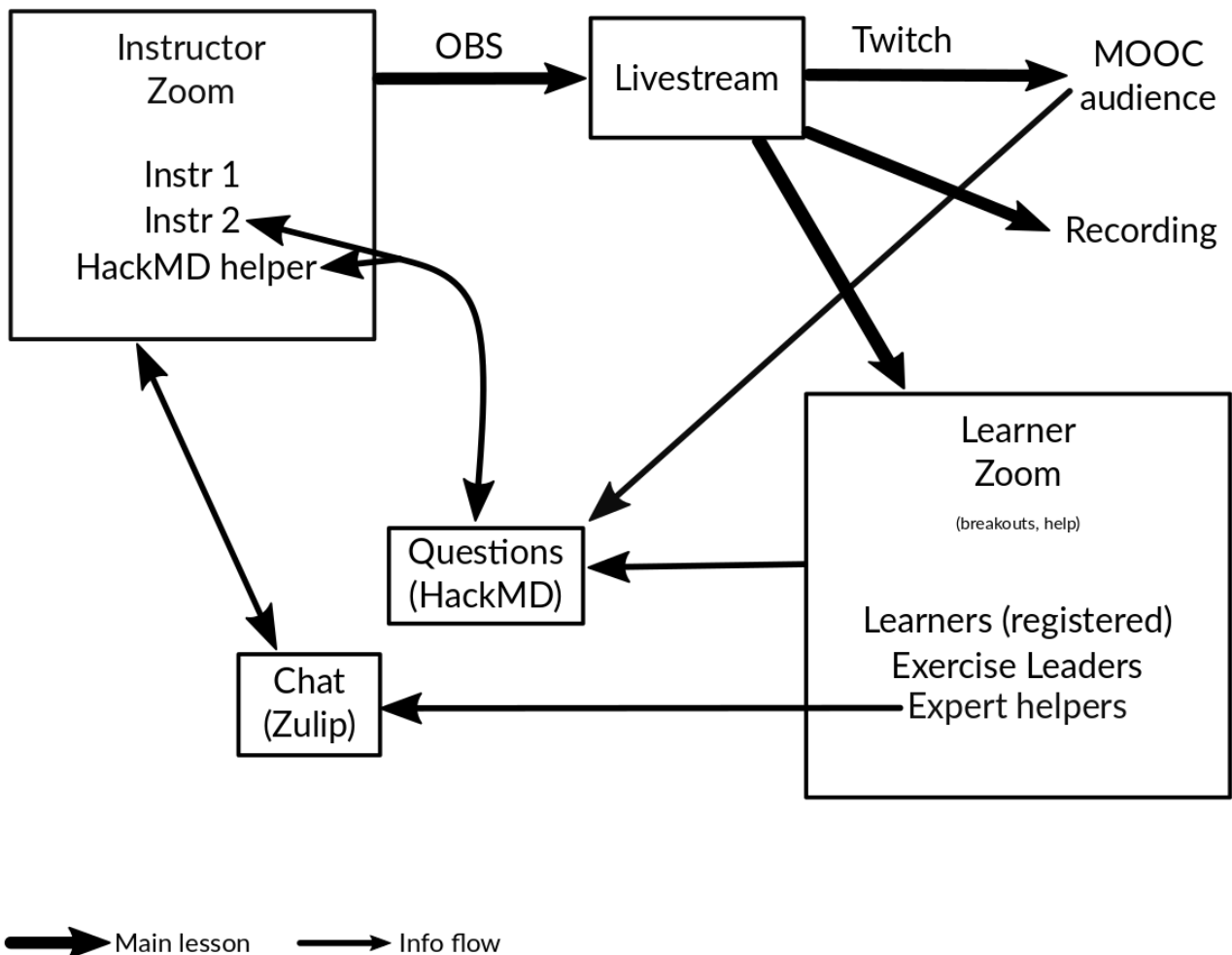
- Reach a near-unlimited number of people
- Fully embrace online tools for interaction, instead of asking people to speak up. This equalizes the participation for shy or passive participants.
- By being large, be more efficient and use the extra resources for meaningful interactions in small groups.

Primary articles

- CodeRefinery MOOC strategy: <https://coderefinery.github.io/manuals/coderefinery-mooc/>
- Intro to livestream teaching: <https://coderefinery.github.io/manuals/livestream-teaching/>
- Broadcaster role description (hints on the actual tools): <https://coderefinery.github.io/manuals/broadcaster/>
- OBS (open broadcaster software) and livestream crash course: <https://coderefinery.github.io/manuals/obs/>

Summary

- There are actually three levels here.
 - In-person
 - Online meeting
 - Online livestream



The general presence and information flow within the MOOC strategy.

- CodeRefinery livestreams via Twitch, but Twitch is not an essential aspect.
- We can invite anyone in the world, no risk of disruptions from trolls.
- This has enabled us to fully embrace strategies such as HackMD and co-teaching.
 - While we tried these in-person, they didn't work well since the loud, extroverted people would dominate.

Tech details

- We stream by using OBS to capture a Zoom meeting. We can switch between a gallery view, screenshare, and mixed.
- Dedicated instructor Zoom meeting - no learners. Thus, no chance of privacy violations.
 - Learners can attend different ways: a) independently online b) in-person breakout room c) Zoom breakout rooms.
- We don't have time to get into details here... see the linked documents and also join us for in-person experience while we improve our materials more.

Exercises

🔧 (advanced) Set up and install OBS as a livestreaming tool.

This exercise is open-ended.

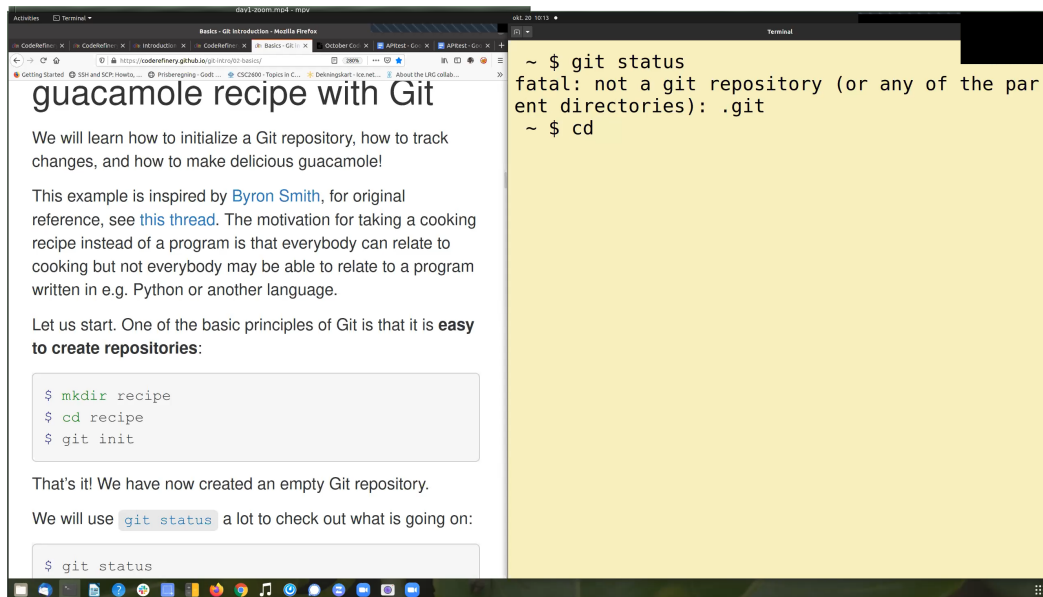
Instructor tech setup

! Keypoints

- Screenshare: **portrait layout** instead of sharing entire screen
- Prompt: **adjust your prompt** to make commands easy to read
- **Readability** and beauty is important: adjust your setup for your audience
- **Share the history** of your commands
- Get set up a **few days in advance** and get an outside reviewer

Screenshare

Compare the following two screen-shares (you can find many more in [the coderefinery manuals](#)):



FullHD.

The screenshot shows a web browser window at the top with the title 'Array jobs - Aalto scientific computing' and a URL 'https://scicomp.aalto.fi/...'. Below the browser is a terminal window. The terminal shows the creation of a file named 'array_example.sh' with the following content:

```
#!/bin/bash
#SBATCH --time=00:15:00
#SBATCH --nodes=2000
#SBATCH --output=array_example_%a.out
#SBATCH --array=0-15

# You may put the commands below:

# Job step
srun echo "I am array task number" $SLURM_ARRAY_TASK_ID
```

Below the script content, the terminal shows the submission of the job to Slurm with the command 'sbatch array_example.sh'. The output shows the job ID 5308576. Then, the command 'sbatch array_example.sh' is run again, and the output shows the job ID 5308576. Finally, the command 'sbatch array_example.sh' is run again, and the output shows the job ID 5308576.

Portrait, latest proposed best practices.

Share a portrait layout (left or right half of your screen) instead of sharing entire screen.


Motivation:

- This makes it easier for you to look at some other notes or to coordinate with other instructors at the same time without distracting with too much information.
- This makes it possible for participants to have something else open in the other screen half: terminal or browser or notebook.

Adjust your prompt/configuration/colors

Adjust your prompt to make commands easy to read. What looks good for your own work (many terminals, small font, extensions, shortcuts, color scheme) is usually not the best for other people watching you. Adjust your setup for your audience. Remove distractions.

You need to spend some time getting set up before you teach. 10 minutes before the session starts is typically too late. Get set up a few days in advance and get an outside reviewer.

 Should instructors be forced to have a consistent screenshare?

There are pros and cons to all instructors using the same screenshare and prompt.

- What are the advantages?
- What are the opportunities of instructors showing different setups?
- How does it depend on the lesson and the experience of learners?

Share the history of your commands

Even if you type slowly it is almost impossible to follow every command. We all get distracted or want to read up on something or while following a command fails on the learner laptop while instructor gets no error.

Add pauses and share the commands that you have typed so that one can catch up.

More examples and how to set it up

- Generally making your screen look good (prompt, remove distractions, command line history, etc): <https://coderefinery.github.io/manuals/instructor-tech-setup/>
- Online setup for screen-shares: <https://coderefinery.github.io/manuals/instructor-tech-online/>

Exercises

Evaluate screen captures

Evaluate screenshots within the [instructor tech setup](#) lesson. Use the collaborative document to make a list of the trade-offs of each one. Which one do you prefer? Which are useful in each situation?

Set up your own environment

Set up your screen to teach something. Get some feedback from another learner. We will discuss among the class.

Video recording

Keypoints

- We don't expect many people to watch the recording from scratch later (but some do) (some might look afterwards a few pieces, cmp. reading a book vs look ing sth up from a book)
- Learners getting an "instant replay" to review, or to **make up for a lost day**, is great.
- Privacy is more important than any other factor
- **Recording only works if privacy is guaranteed and effort is low.** This is only possible with the **instructor-audience split setup** of livestreaming.

We try to release videos on the same day

Video recordings could be useful for people attending a course later, but also are (perhaps more) useful for **immediate review and catching up after missing a day in a workshop**.

For this, they need to be released immediately, within a few hours of the workshop (see [Video editing](#)). CodeRefinery can do this.

For the videos to be released soon we need to make sure we guarantee privacy of learners (see below). Our livestream setup makes the privacy guarantee easy at the cost of separating instructors and learners into different video rooms.

Good preparation and documentation helps to make video recording and video editing easier.

Privacy is more important than any other factor

If we can't guarantee privacy, we can't release videos at all.

Some events add a disclaimers such as "if you don't want to appear in a recording, leave your video off and don't say anything". We would prefer not to do this, since:

- we *know* accidents happen (especially when coming back from breakout rooms)
- it creates an incentive to not interact by voice/video
- it could pressure participants to not object in order "to not be difficult"

Livestreaming solves this for us:

- By separating the instruction from the audience audio/video, there is no privacy risk in the raw recording. They could be released or shared unprocessed.
- Recording with Zoom in a large meeting with all the instructors and learners is simple, but not good for privacy: there are always mistakes, reviewing takes too long.
- Livestream platforms also provide instant recordings of the whole stream, and some make instant replays possible. This could remove the need for making our own videos, since one of the most important cases is this instant replay idea.

Broadcasting role and setup

In the live-streaming setup, the Broadcasting role is central to video recording.

Broadcaster description (most is not directly about recording):

<https://coderefinery.github.io/manuals/broadcaster/>

Video editing

Video recordings could be useful for people attending a course later, but also are (perhaps more) useful for **immediate review and catching up after missing a day in a workshop**. For this, they need to be released immediately, within a few hours of the workshop.

CodeRefinery can do this.

Hypothesis: videos must be processed the same evening as they were recorded, otherwise (it may never happen) or (it's too late to be useful). To do that, we have to make processing *good enough* (not perfect) and *fast* and *distributeable*

Primary articles

- Video editor role description: <https://coderefinery.github.io/manuals/video-editor/>
- ffmpeg-editlist: the primary tool: <https://github.com/coderefinery/ffmpeg-editlist>
 - Example YAML editlists: <https://github.com/AaltoSciComp/video-editlists-asc>

Summary

- Basic principle: privacy is more important than any other factor. If we can't guarantee privacy, we can't release videos at all.
 - Disclaimers such as "if you don't want to appear in a recording, leave your video off and don't say anything", since a) accidents happen especially when coming back from breakout rooms. b) it creates an incentive to not interact or participate in the course.
- Livestreaming is important here: by separating the instruction from the audience audio/video, there is no privacy risk in the raw recording. They could be released or shared unprocessed.
- Our overall priorities
 1. No learner (or anyone not staff) video, audio, names, etc. are present in the recordings.
 2. Good descriptions.
 3. Removing breaks and other dead time.
 4. Splitting videos into useful chunks (e.g. per-episode), perhaps equal with the next one:
 5. Good Table of Contents information so learners can jump to the right spots (this also helps with "good description".)
- [ffmpeg-editlist](#) allows us to define an edit in a text file (crowdsourceable on Github), and then generate videos very quickly.

Exercises

Exercise A

These exercises will take you through the whole sequence.

Editing-1: Get your sample video

Download a sample video:

- Video (raw): <http://users.aalto.fi/~darstr1/sample-video/ffmpeg-editlist-demo-kickstart-2023.mkv>
- Whisper subtitles (of raw video): <http://users.aalto.fi/~darstr1/sample-video/ffmpeg-editlist-demo-kickstart-2023.srt>
- [Schedule of workshop](#) (day 1, 11:35–12:25) - used for making the descriptions.

Editing-2: Run Whisper to generate raw subtitles and test video.

First off, install Whisper and generate the base subtitles, based on the. Since this is probably too much to expect for a short lesson, they are provided for you (above), but if you want you can try using Whisper, or generating the subtitles some other way.

You can start generating subtitles now, while you do the next steps, so that they are ready by the time you are ready to apply the editlist. `ffmpeg-editlist` can also slice up the subtitles from the main video to make subtitles for each segment you cut out.

Whisper is left as an exercise to the reader.

✓ Solution

Example Whisper command:

```
whisper --device cuda --output_format srt --initial_prompt="Welcome to  
CodeRefinery day four." --lang en --condition_on_previous_text False INPUT.mkv
```

An initial prompt like this make Whisper more likely to output full sentences, instead of a stream of words with no punctuation.

Editing-3: Create the basic editlist.yaml file

Install `ffmpeg-editlist` and try to follow its instructions, to create an edit with these features:

- The input definition.
- Two output sections: the “Intro to the course”, and “From data storage to your science” talks (Remember it said the recording started at 11:35... look at the schedule for hints on when it might start!). This should have a start/end timestamp from the *original* video.

A basic example:

```
- input: day1-raw.mkv

# This is the output from one section. Your result should have two of these
# sections.
- output: part1.mkv
  title: something
  description: >-
    some long
    description of the
    segment
  editlist:
    - start: 10:00      # start timestamp of the section, in *original* video
    - end: 20:00       # end timestamp of the section, in the *original* video
```

✓ Solution

This is an excerpt from our [actual editlist file of this course](#)

```
- input: day1-obs.mkv

- output: day1-intro.mkv
  title: 1.2 Introduction
  description: >-
    General introduction to the workshop.

    https://scicomp.aalto.fi/training/kickstart/intro/

  editlist:
    - start: 00:24:10
    - end: 00:37:31

- output: day1-from-data-storage-to-your-science.mkv
  title: "1.3 From data storage to your science"
  description: >-
    Data is how most computational work starts, whether it is
    externally collected, simulation code, or generated. And these
    days, you can work on data even remotely, and these workflows
    aren't obvious. We discuss how data storage choices lead to
    computational workflows.

    https://hackmd.io/@AaltoSciComp/SciCompIntro

  editlist:
    - start: 00:37:43
    - end: 00:50:05
```

Discussion: what makes a video easy to edit?

- Clear speaking and have high audio quality.
- For subtitle generation: Separate sentences cleanly, otherwise it gets in a “stream of words” instead of “punctuated sentences” mode.
- Clearly screen-sharing the place you are at, including section name.
- Clear transitions, “OK, now let’s move on to the next lesson, LESSON-NAME. Going back to the main page, we see it here.”
- Clearly indicate where the transitions are
- Hover mouse cursor over the area you are currently talking about.
- Scroll screen when you move on to a new topic.
- Accurate course webpage and sticking to the schedule

All of these are also good for learners. By editing videos, you become an advocate for good teaching overall.

Editing-4: Run ffmpeg-editlist

Install ffmpeg-editlist: `pip install ffmpeg-editlist[srt]` (you may want to use a virtual environment, but these are very minimal dependencies).

The `ffmpeg` command line tool must be available in your `PATH`.

✓ Solution

It can be run with (where `.` is the directory containing the input files):

```
$ ffmpeg-editlist editlist.yaml .
```

Just running like this is quick and works, but the stream may be garbled in the first few seconds (because it's missing a key frame). (A future exercise will go over fixing this. Basically, add the `--reencode` option, which re-encodes the video (this is **slow**). Don't do it yet.

Look at the `.info.txt` files that come out.

Editing-5: Add more features

- Several chapter definitions.(re-run and you should see a `.info.txt` file also generated). Video chapter definitions are timestamps of the *original* video, that get translated to timestamps of the *output* video.

```
- output: part1.mkv
editlist:
- start: 10:00
- -: Introduction    # <-- New, `:` means "at start time"
- 10:45: Part 1      # <-- New
- 15:00: Part 2      # <-- New
- end: 20:00
```

Look at the `.info.txt` files that come out now. What is new in it?

- Add in “workshop title”, “workshop description”, and see the `.info.txt` files that come out now. This is ready for copy-pasting into a YouTube description (first line is the title, rest is the description).

Look at the new `.info.txt` files. What is new?

✓ Solution

- This course actually didn't have chapters for the first day sessions, but you can [see chapters for day 2 here](#), for example.

- [Example of the workshop description for this course](#)
- Example info.txt file for the general introduction to the course. The part after the `-----` is the workshop description.

```
1.2 Introduction - HPC/SciComp Kickstart summer 2023

General introduction to the workshop.

https://scicomp.aalto.fi/training/kickstart/intro/

00:00 Begin introduction      <-- Invented for the exercise demo, not real
03:25 Ways to attend         <-- Invented for the exercise demo, not real
07:12 What if you get lost    <-- Invented for the exercise demo, not real

-----

This is part of the Aalto Scientific Computing "Getting
started with Scientific Computing and HPC Kickstart" 2023
workshop. The videos are available to everyone, but may be
most useful to the people who attended the workshop and want
to review later.

Playlist:
https://www.youtube.com/playlist?list=PLZLVmS9rf3nMKR2jMglaN4su3ojWtWMVw

Workshop webpage:
https://scicomp.aalto.fi/training/scip/kickstart-2023/

Aalto Scientific Computing: https://scicomp.aalto.fi/
```

Editing-6: Subtitles

Re-run `ffmpeg-editlist` with the `--srt` option (you have to install it with `pip install ffmpeg-editlist[srt]` to pull in the necessary dependency). Notice how `.srt` files come out now.

Use some subtitle editor to edit the *original* subtitle file, to fix up any transcription mistakes you may find. You could edit directly, use `subtitle-editor` on Linux, or find some other tool.

What do you learn from editing the subtitles?

✓ Solution

..code-block:

```
$ ffmpeg-editlist --srt editlist.yaml
```

There should now be a `.srt` file also generated. It generated by finding the `.srt` of the original video, and cutting it the same way it cuts the video. Look and you see it aligns with the original.

This means that someone could have been working on fixing the Whisper subtitles while someone else was doing the yaml-editing.

Editing-6: Subtitles

Re-run `ffmpeg-editlist` with the `--srt` option (you have to install it with `pip install ffmpeg-editlist[srt]` to pull in the necessary dependency). Notice how `.srt` files come out now.

Use some subtitle editor to edit the *original* subtitle file, to fix up any transcription mistakes you may find. You could edit directly, use `subtitle-editor` on Linux, or find some other tool.

What do you learn from editing the subtitles?

Editing-7: Generate the final output file.

- Run `ffmpeg-editlist` with the `--reencode` option: this re-encodes the video and makes sure that there is no black point at the start.
- If you re-run with `--check`, it won't output a new video file, but it *will* re-output the `.info.txt` and `.srt` files. This is useful when you adjust descriptions or chapters.

Discussion: how to distribute this?

Create a flowchat of all the parts that need to be done, and which parts can be done in parallel. Don't forget things that you might need to do before the workshop starts.

How hard was this editing? Was it worth it?

Exercise B

This is similar to the above but more brief and not on a real example video.

Use `ffmpeg-editlist` to edit this sample video

Prerequisites: `ffmpeg` must be installed on your computer outside of Python. Be able to install `ffmpeg-editlist`. This is simple in a Python virtual environment, but if not the only dependency is `PyYAML`.

- Download the sample video: <http://users.aalto.fi/~darstr1/sample-video/sample-video-to-edit.raw.mkv>
- Copy a sample editlist YAML
- Modify it to cut out the dead time at the beginning and the end.
- If desired, add a description and table-of-contents to the video.
- Run ffmpeg-editlist to produce a processed video.

✓ Solution

```
- input: sample-video-to-edit.raw.mkv
- output: sample-video-to-edit.processed.mkv
description: >
editlist:
  - start: 00:16
  - 00:15: demonstration
  - 00:20: discussion
  - stop: 00:25
```

```
$ ffmpeg-editlist editlist.yaml video/ -o video/
```

Along with the processed video, we get `sample-video-to-edit.processed.mkv.info.txt` :

```
This is a sample video
```

```
00:00 Demonstration
00:04 Discussion
```

See also

- ffmpeg-editlist demo: <https://www.youtube.com/watch?v=thvMNTBJg2Y>
- Full demo of producing videos (everything in these exercises):
https://www.youtube.com/watch?v=_CoBNe-n2Ak
- Example YAML editlists: <https://github.com/AaltoSciComp/video-editlists-asc>

Distributed workshop organization

📌 Keypoints

- If another university/organization wants to run the course, they can easily join our existing course at little cost/risk to them and to us.
- Each organization that joins provides a great benefit to us (helpers, instructors).
- They can reserve an in-person breakout room and provide mentors while watching the livestream: A great experience for their audience.

It is easier to join and follow than to start or lead

Organizing a course is time intensive and so is developing a course. Many courses given in country A are also relevant in other countries and instead of developing and possibly re-inventing courses independently in different places, it is so much more rewarding to collaborate and share.

In our courses we wish and try to make it easier for teams and organizations and universities to join, at little risk and cost to them, possibly for a short period of time, possibly only as observer. Later observers can progress to exercise leads, expert helpers, instructors, and co-organizers.

One model that we have successfully tried is that the course is delivered via live-stream to all partners and all the partner needs to do is to reserve a room and motivate few people to help local learners as exercise leads during exercise sessions which can happen in-person.

Lessons learned from organizing larger workshops

- **One person is needed to coordinate the registration process** and this person should not have teaching duties in addition to this role.
- Generally it helps to have **well-defined roles** (see [Workshop roles](#)).

Why teach together?

We usually say so much about the value of collaboration. Despite us saying this, our teaching is still far too alone. Covid-19 gave us a kick that reduced our barriers and led to lasting changes in how we taught (eventually leading to this very course).

Open Science / FAIR data is heavily emphasised these days. But let's add a "C" to FAIR: "collaborative". Instead of people doing their own thing and releasing, develop, iterate, and maintain **collaboratively**.

Ways to teach together

- Develop materials together - avoid duplication.
- [Team teaching](#)
- Extensive use of helpers and team leaders.
- HackMD for parallel and mass answers.

Advantages

- You need to teach anyway, less effort if you combine.
- If done right, minimal extra effort for others to receive benefit (+ you get publicity).
- Many of the previously presented teaching strategies work best in large courses - this makes the course more engaging than a small event with minimal interaction.
- More engaging for the audience.

- Easier on-boarding of new instructors (less “scary” to teach a new course with other instructors).

Challenges and disadvantages

- Coordination
 - Finding suitable partners with the same vision
 - Coordination efforts (if others don’t understand the vision).
- Materials
 - May not be perfectly tuned to your own audience
 - May not iterate as fast as you need
- Co-teaching
 - Difficulty in finding co-teachers
 - Required effort of syncing among staff
 - It might revert to independent teaching if you aren’t careful.
- HackMD
 - Can possibly overload both student and teacher.

Exercises

What similarities do we have?

Using HackMD, make two lists:

- What courses do you think your local community would benefit from, which you don’t currently have? +1 other people’s items which are also relevant to you.
- Which courses are you thinking of preparing for your local community? +1 other people’s items which you would be interested in helping out with.

See also

- [Collaboration models](#)

Collaboration models

Model: CodeRefinery

- Before Covid-19, workshops were physically around the Nordics, instructors would travel (or already be there).
 - Maximum size: ~40 people
 - High workload per person
- After several small scaling attempts, now we have:
 - Two large workshops per year - livestream format
 - Combined organization efforts
 - Instructors from each location - on average two lessons taught.
 - Locations with staff can have **local breakout rooms**: physical place to help during exercises.

- Others in the world can register and interact using HackMD, but no promises of help.
- Content still available to anyone in the world: live + instant replay.
- Course page and material: <https://coderefinery.github.io/2022-03-22-workshop/>

Model: Python for Scientific Computing

- Aalto Scientific Computing wanted to host a course, **Python for Scientific Computing**
- ASC came up with initial vision and announced it
- ASC hosted an open initial meeting, inviting any interested organizers or instructors
- We went over the plan and refined the topics and schedule. We also decided things such as the date, organizers, and instructors for each lesson.
- Registration was open to everyone in the world, non-Nordic participants could watch via livestream.
- People prepared their parts and came together and presented. Organizers kept everything on track.
- **Compared to the amount of effort each person put in, the results were great.**
- A 2021 version also happened and was even larger.
- Course page: <https://scicomp.aalto.fi/training/scip/python-for-scicomp/>
- Material: <https://aaltoscicomp.github.io/python-for-scicomp/>

Exercises

List successes and failures in collaborative teaching

Using HackMD, list some successes and failures in collaborative teaching that you have experienced.

Recommendations for co-teaching

If you have experience with co-teaching, what approach/technique/trick can you recommend a colleague who would like to try co-teaching for the first time?

Workshop roles

Running massive workshops requires a lot of people. We have a variety of roles with different levels of support.

As usual, roles are a plan, and a plans are made to be updated.

Primary articles

- Workshop roles: <https://coderefinery.github.io/manuals/roles-overview/>

Summary

Lower levels mean “top level sometimes split into some of these sub-roles”.

- **Instructor coordinator:** coordinates schedule and instructors
 - **Instructor:** Teaches along with a co-teacher.
 - **Expert helper:** Spare person, usually watching HackMD but also rotates among breakout rooms. Often instructs some lessons.
 - **Director:** Manages the flow of the schedule during the workshop, introduces each lesson, etc. (often the instructor coordinator)
 - **Broadcaster:** Manages the livestreaming
 - **Video editor:** Edits and publishes videos the day of the workshop.
- **Registration coordinator:** coordinates registration, helpers, and breakout rooms.
 - **Exercise leader coordinator:** Onboards exercise leaders
 - **Host:** Manages the learner breakout rooms, learner questions, etc. (often the registration coordinator)
 - **Advertisement coordinator:** Advertises and outreaches
- Under both registration coordinator and instructor coordinator
 - **HackMD manager:** always watches and formats HackMD and publishes it same-day. “Eyes on the ground” via HackMD and chat and quickly communicates important information to instructor and registration coordinators.
- **Learner:** attends and learns
- **Exercise leader:** serves as a guide to the team, receives small amount of training before the workshop.

Exercises

How many people teach in your workshops?

- Using HackMD, make a histogram of how many (instructors + organizers) you typically have in your workshops.
- List some of the common roles you have used.

See also

(none yet)

Why are computers hard?

Most of the time, when teaching, our difficulty is not what you expect.

Initial reading

Read the following:

How to help someone use a computer, by Phil Agre

<https://www.librarian.net/stax/4965/how-to-help-someone-use-a-computer-by-phil-agre/>

Of each of the points made, how many are related to:

- The computing itself
- The user interface
- The ability of the user to work in the computing environment
- Something else

Usability

As said in the text above:

Most user interfaces are terrible. When people make mistakes it's usually the fault of the interface. You've forgotten how many ways you've learned to adapt to bad interfaces. You've forgotten how many things you once assumed that the interface would be able to do for you.

Deep abstraction layers

Most technology is built on abstraction layers, for good reason. They help simplify implementation and understanding.

Exercise

Think of a tool or technology that is easy to understand and use if you understand the underlying abstraction layers, but is almost impossible otherwise.

Conclusion: what are we teaching, then?

As teachers of computing, we fill a critical role that is more determined by our audience, than the technology we are teaching.

See also

(none yet)

Diversity and inclusion

When we are teaching, we fill a critical role in between people and technology. We are not un-involved technical staff teaching asocial topics, but we are very connected to the social context of the tools. Unfortunately, there are major demographic imbalances in most computing.

Primary articles

- Several sections from the Carpentries Instructor Training: [Motivation and demotivation](#) and [Equity, inclusion, accessibility](#).
- Exercise leaders [During the workshop](#)

Support services vs diversity

Study this presentation by Richard Darst:

- Watch: <https://www.youtube.com/watch?v=z1VS1wleN-o>
- Or read: https://docs.google.com/presentation/d/e/2PACX-1vQkzOMFI5SFnx69D4qRq_3N-mtcv7GUbPZd4A6UYpEKZUMGK0FL5W0RsSe6Alzxv1nE635Yl1GpyCKk/pub

Summary:

- Computing is hard and quite often our ability to learn quickly is connected to our social group.
- Good technical services serve the role of mentors
- This mentorship helps to equalize
- Our entire system of research should be configured for more equality. Modernization usually takes it the other way.

Exercises

 Reflect on how your job can be defined to promote diversity.

What are some (possibly surprising) ways that your job promote diversity and equality among people of different backgrounds?

See also

(none yet)

Placeholder

Lesson development with version control

So, we want to practically share. We have these minimum requirements:

- Someone can get the preferred form of modification, to improve without limitation.
- It is trivial to track differences and send the changes back to the source, with little cost to the original maintainer.

Especially with the second of these, version control in an online platform seems to be the only reasonable option.

Version control and static site generators

CodeRefinery and the Carpentries use git and Github to develop lessons. The general procedure is this:

- Version control to store the raw files in text format
- A static website compiler to convert them to HTML files
- Serving to the public via Github Pages (but this could be replaced with other systems)

This allows for true collaborative development and community contributions.

Carpentries uses a custom website compiler based on R, CodeRefinery uses the Sphinx documentation engine which is used in many different projects for documentation.

CodeRefinery's use of a standard tool allows us to build on a huge and growing ecosystem of extensions and themes, and promotes local reuse.

The exact static website generator used isn't so important, as long as some form of version control is used.

A open-source license is the last bit to consider: without a license, it can't be reused and passed on, and there is little incentive for someone to contribute.

CodeRefinery lesson tools

CodeRefinery uses the following tools to actually make its lessons right now:

- [Sphinx](#) (a common documentation generator, widely used in open source projects in general)
- The [sphinx-lesson](#), which is more of a small collection of other extensions than new development itself.
- Github for hosting lessons: <https://github.com/coderefinery/>
- Github Actions and Github Pages for building and web serving our lessons.

Exercises

Contribute to a sample lesson

- Open this very lesson in GitHub (it uses the same format as typical CodeRefinery lessons): <https://github.com/coderefinery/community-teaching/>
- Browse the files and understand the general idea. Check out at least these and use HackMD to record their functions:
 - `.github/workflows/sphinx.yml`
 - `content/conf.py`
 - `content/index.rst`
 - `content/lessons-with-version-control.rst`
- If you want, try to make a pull request to this lesson. It doesn't have to have any significant content, it can be a pure test pull request.

(advanced) Create your own lesson

Use the [sphinx-lesson-template](#) to create a new lesson of your choice. Alternatively, use the current Carpentries system, or some other system of your choice.

Recommendations and lessons learned

- Convert feedback about lessons and suggestions for improvements into *issues* so that these don't get lost.
- Make your lesson citable: get a DOI.
- Credit contributors (not only Git commits).
- Instructor guide is essential for new instructors.
- Lesson changes should be accompanied with instructor guide changes (it's like a documentation for the lesson material).
- Apply and validate [Backwards lesson design](#) again and again.
- Make it possible to try out new ideas (by making the lesson branch-able).
- Before making larger changes, talk with somebody and discuss these changes.
- For substantial changes we recommend to first open an issue and describe your idea and collect feedback before you start with an extensive rewrite.
- For things still under construction, open a draft pull request to collect feedback and to signal to others what you are working on.

See also

- [CodeRefinery git-intro](#)
- [CodeRefinery git-collaborative](#)
- [Carpentries lesson development](#)

Backwards lesson design

It happens far too often: someone creates a lesson, but they think about what is interesting to them, not what is important for the learners. In fact, an earlier version of this instructor training had this very issue.

It is critical to backwards design almost any piece of communication, especially something as widespread as teaching.

The approach

- You don't think about how to do something and try to explain it.
- Avoid the typical approach *"I want to show a number of things which I think are cool about tool X - how do I press these into 90 minutes?"*
- Instead, you start defining your target audience by answering to questions such **What is the expected educational level of my audience?**, **Have they been already exposed to the technologies I am planning to teach?**, **What tools do they already use?**, **What are the main issues they are currently experiencing?**. It is important to discuss these points with a group of colleagues, preferably from diverse backgrounds and institutions to reduce biases. Once you clarified your target audience, it is useful to create **learner personas**;

that will help you during the development process by providing concrete examples of potential learners showing up at your workshops. For each **learner personas**, try to think of what is **useful to them**: “What do they **need** to [remember/understand/apply/analyze/evaluate/create](#)?”. Asking and answering to these questions will allow you to define the background knowledge (starting points) and goals (end points) of your learners. Then, you create a sequence of exercises which test incrementally progressing tasks and acquisition of the new skills (from starting to end points).

- Then, you write the minimum amount of material to teach the gap between exercises.

The process

As described in “[A lesson design process](#)” in the book [Teaching Tech Together](#):

1. Understand your learners
2. Brainstorm rough ideas
3. Create an summative assessment to know your overall goal
 - CodeRefinery translation: think of the things your learners will be able to do at the end of the lesson. Think simple! The simpler the better. Think of three main points they will remember, of which maybe one or two are a concrete skill.
4. Create formative assessments to go from the starting point to this.
 - CodeRefinery translation: think of some engaging and active exercises.
5. Order the formative assessments (exercises) into a reasonable order.
6. Write just enough material to get from one assessment (exercise) to another.
7. Describe the course so the learners know if it is relevant to them.

We can’t emphasize enough how important it is to **know your end state and keep it simple**.

Example: designing an HPC Carpentry lesson

Let’s take as an example the [HPC Carpentry lesson](#)

Target audience

- What is the expected educational level of my audience?
 - A PhD student, postdoc or young researcher.
- Have they been already exposed to the technologies I am planning to teach?
 - The word **HPC** is not new to them and they may have already used an HPC but are still not capable of giving a proper definition of HPC. In addition, we do not expect them to know much about parallelism and they cannot make any distinction between various available parallelism paradigms.
- What tools do they already use?
 - serial codes, multi-threaded codes, data parallelism; usually out-of-the-box tools.
 - they may have tried to “scale” their code (multiprocessing, threading, GPUs) with more or less success.
- What are the main issues they are currently experiencing?

- they cannot solve their problems either because they would like to run the same code but with many different datasets or because their problem is larger (more computations/memory).
- most of the time they know their codes can run on HPC (from the documentation) but never really had the opportunity to try it out.
- Very few will have their own codes where they may have tried different things to speed it up (threading, task parallelism) but have no clear strategy.

Learner persona

- Sonya is a 1st year PhD student: she recently moved to Oslo and joined the Computational and Systems Neuroscience group. She will be using the [NEST](#), a simulator for spiking neural network model. She used NEST during her master thesis but on her small cluster: **she never used an HPC resource** and is really excited about it.
- Robert is a field ecologist who obtained his PhD 6 months ago. He is now working on a new project with Climate scientists and as a consequence will need to run global climate models. He is **not very familiar with command line** even though he attended a Software Carpentry workshop and the idea to use HPC is a bit terrifying. He knows that he will get support from his team who has extensive experience with HPC but would like to become more independent and be able to **run his own simulations** (rather than copying existing cases).
- Jessica is a postdoc working on a project that investigates numerically the complex dynamics arising at the tip of a fluid-driven rupture. Fluid dynamics will be computed by a finite element method solving the compressible Navier-Stokes equations on a moving mesh. She **uses a code she has developed** during her PhD and that is based on existing libraries. She has mostly ran it on a local desktop; her work during her PhD was very limited due to the lack of computing resources and she is now very keen is **moving to HPC**; she knows that it will requires some work, in particular to parallelize her code. This HPC training will be her first experience with HPC.

Learning outcomes

- Understand the difference between HPCs and other local/remote machines
- Understand the notion of core, nodes, cluster, shared/distributed memory, etc.
- Understand the notion of login nodes.
- Understand the need for a scheduler and how to use it appropriately
- Understand why optimising I/O is important on HPC and how to best use HPC filesystems
- Understand the need to parallelize (or use existing parallel) codes and in which cases HPCs is a must (when communications is required)
- Understand how to get your code ready to use on HPC (access to libraries, installation of your own libraries/software, etc.)
- Understand that an HPC is an operational machine and is not meant for developing codes.

Exercises

- Get basic information such the number of CPUs, memory from your laptop and try to do the same on a HPC. Discuss outcomes.
- Try to create files on the different filesystems on your HPC resource and access them.
- Create different types of job scripts, submit and check outputs.
- Make a concrete example to run a specific software on your HPC (something like GROMACS).

Exercises

Backwards-design a lesson/topic

Choose a simple lesson topic and apply backwards lesson design. You won't get all the way through, but come up with a logical progression of exercises.

The section you pick should require **screen sharing** and be of some **follow-along task** (preferably using a shell).

Some suggestions:

- Regular expressions
- Making papers in LaTeX
- Making figures in your favorite programming language
- Linux shell basics
- Something non-technical, such as painting a room
- An instructor training for CodeRefinery
- Some aspect from an already existing lesson
- [Introduction to high-performance computing](#) (or an episode therein)
- [Unix shell in a HPC context](#) (or an episode therein)
- A lesson you always wanted to teach

Possibilities for Carpentries

Keypoints

- Carpentries is currently designed around small workshops, so many of these ideas can't directly apply
- Yet **many of these tools and also team teaching can still be used**
- You can run your own breakout room for any of our workshops
- Join as observer if you want to see our workshop organization and tools in action

CodeRefinery workshops can be a good progression step after taking a Carpentries workshop.

Helping out at a CodeRefinery workshop as exercise lead can be a good progression step for those who have helped out at a Carpentries workshop or who have taken the Carpentries instructor training.

CodeRefinery's plans

! Keypoints

- We are continuing to focus on online-first with local breakout rooms
- We welcome people joining us, either individually or as an organization
- Still interested in collaboration with Carpentries
- We need to become better at marketing and outreach

Biggest open problems

- How to give helpers and contributors more credit and visibility
- How to promote/engage new members
- What are we? Non-profit? Institution collaboration network? Selling services?
- Funding and sustainability (but we have ideas: collaboration network)

How you can join

Individual level:

- Join [CodeRefinery chat](#)
- Lead a team, co-teach, or help organize a workshop
- Generally provide marketing and outreach

Organization level:

- Have your organization join CodeRefinery
- Officially co-advertise and co-teach workshops
- Run local breakout rooms and join a workshop as a team
- Send an observer to a workshop

Aside: Nordic-RSE (research software engineers)

- [Nordic-RSE online unconference 2022](#)
- [Bi-weekly meetings](#)

Other resources

- The future of teaching" (35 min content only, 45 min with Q&A, or 15 min reading). [Watch it on YouTube](#) or [read it](#) if you prefer.
- [The old "CodeRefinery instructor training" program](#): this is replaced by what you are reading now.

- [CodeRefinery manuals](#)
- Our gradual pathway to instructor from the manuals: [helper intro](#) and [instructor intro](#).
- [Carpentries instructor training](#): more intensive, but focused only on teaching.
- [Carpentries curriculum development handbook](#)
- [Teaching Tech Together](#): a book about similar topics by someone involved in Carpentries.
- [Carpentries general organizational documentation](#)
- [How to help someone use a computer, by Phil Agre](#). Summary: Most of our teaching challenge is helping people to overcome bad user interface design.
- [The Science of Learning](#): provides a brief overview of some key evidence-based results in teaching.

Instructor's guide

In a lesson that was further developed, this would include an instructor's guide that mentioned:

- Background of why the course is how it is
- Recommendations of teaching it
- Known pitfalls of teaching it, so that other instructors can avoid them
- Possibly how to contribute, long-term plans, etc.

For example, see [git-intro's instructor guide](#).

Exercise list

This is a list of all exercises and solutions in this lesson, mainly as a reference for helpers and instructors. This list is automatically generated from all of the other pages in the lesson. Any single teaching event will probably cover only a subset of these, depending on their interests.

CodeRefinery teaching philosophies

Ice-breaker in groups (20 minutes)

- Share your approach to teaching and your teaching philosophy with your group.
- Please share your tricks and solutions in the live document for others.

Additional ice-breaker questions:

- What is your motivation for taking this training?
 - How structured or informal are your own teaching needs?
 - What difference do you notice between the teaching what we (also Carpentries) do and traditional academic teaching?
 - What other skills need to be taught, but academic teaching isn't the right setting?
-

Here CodeRefinery instructors share their training philosophy to show that we all have different teaching styles and how these differences are beneficial to CodeRefinery workshops.

It is important to explain how much we value individuals and that there is not one way to teach but as many ways as individuals. We want to help each other to find the way that is best for each of us.

Video recordings

Recently we have recorded some of the below as videos:

<https://www.youtube.com/playlist?list=PLpLbIYHCzJAAHF89P-GCjEXWC8CF-7nhX>

Anne Fouilloux

I regularly teach Carpentries workshops so I try to apply what I have learnt to CodeRefinery workshops. However, I know our target audience is very much different and that I need to adapt my teaching style. I am still trying to find what works best in which situations and this is why I like so much CodeRefinery workshops. We usually have a wider range of skills and very mixed backgrounds so we usually have to be more careful with the pace and time given for exercises.

Some considerations:

- I spend quite a lot of time reading the CodeRefinery material and practising myself exercises. I particularly like to read the instructor notes just before teaching: they usually highlight important aspects both for preparing and teaching.
- I usually do not show too much in advance the material as I think it prevents asking questions. If you have less competent practitioners in the classroom, they can easily copy-paste to avoid slowing down the entire classroom.
- Ideally, I'd like to give several exercises so anyone can work at its own pace. I find it is important that everybody gets something different from the workshop.
- I love breaks as it gives us an opportunity to discuss with attendees on their research topics. I am especially interested to understand what software they write and how they plan to use what they learn during our workshops.

Bjørn Lindi

My teaching style has changed a bit since I started with CodeRefinery. In the beginning I had this “BLOB” (Binary Large Object) of knowledge and experience that I wanted to convey to the participants. With experience and some help from the Carpentries Instructor training, I have realized I need to serialize the “BLOB”, to be able to share it with others.

In a similar fashion as you would do with a binary large object which you intend to send over the wire, you will need stop signals, check-sums and re-transmissions, when you give a lecture. I have come to appreciate the natural periods/breaks the lessons offers, the questions raised, the errors that appear during type-along and the re-transmission. Co-instructors are good to use for re-transmission or broadening a specific topic.

When I started with CodeRefinery my inclination was to give a lecture. Today I am trying to be a guide during a learning experience, a learning experience which includes me as well. That may sound a bit self-centric, but is in fact the opposite, as I have to be more sensitive to what is going on in the room. The more conscious I am of being a guide, the better lesson.

Tools that I find useful in preparing a lesson is concept maps and Learner Personas, though I have develop to few them.

- [Concept Maps](#)
- [Learner Personas](#)

Thor Wikfeldt

I never want to leave any learner behind and I really don't like seeing confused, blank faces in the classroom. At the same time I sometimes worry about some participants getting bored if a lesson is progressing slowly. This is always a difficult compromise and something I struggle with!

I try to focus on making concepts intuitive, to "make sense" to the learners. Of course this is usually based on how I learned the topic myself and how it makes sense to me.

I try to establish connections between topics: "this thing here is similar to what we saw in the previous lesson where we learned about X..."

Before mastering a lesson by teaching in many times I try to "follow the script". After becoming very familiar with a lesson I start to improvise more and react more dynamically to questions, e.g. by taking a detour to explain a confusing topic more clearly.

What I think I do too often: copy-paste code/text from lesson material. This can leave learners behind - typing out the code and describing it is slower, but more learning takes place. More advanced learners will hopefully "be compensated" by interesting advanced exercises which follow.

Stefan Negru

A lesson is a conversation, it is useful if both the trainer and the trainee are engaged. For that reason I try to have, most of the time, a conversation with the classroom and after we finish parts of a lesson, step back and see how we might use what we learned.

That brings me to another point I follow throughout the lessons, answering questions like:

- How can we apply in practice what we just learned?
- Do you see yourself (the trainee) using that in practice, why or why not?

Most of the times those seem like open-ended questions to the trainees that just learned something new, so I try to find examples, most of the times from personal experience.

Last thing is that analogies are important when I teach, I try to find analogies in order to simplify a convoluted part of a lesson.

Radovan Bast

My teaching changed by 180 degrees after taking the Carpentries instructor training. Before that I used slides, 45 minute lecture blocks, and separate exercise sessions. After the Carpentries instructor training I embraced the interaction, exercises, demos, and typos.

My goal for a lesson is to spark curiosity to try things after the lesson, both for the novices (“This looks like a useful tool, I want to try using it after the workshop.”) and the more experienced participants (“Aha - I did not know you could do this. I wonder whether I can make it work with X.”). I like to start lessons with a question because this makes participants look up from their browsers.

Keeping both the novices and the experts engaged during a lesson can be difficult and offering additional exercises seems to be a good compromise.

For me it is a good sign if there are many questions. I like to encourage questions by asking questions to the participants. But I also try not to go into a rabbit hole when I get a question where only experts will appreciate the answer.

I try to avoid jargon and “war stories” from the professional developers’ perspective or the business world. Most researchers may not relate to them. For examples I always try to use the research context. Avoid “customer”, “production”, also a lot of Agile jargon is hard to relate to.

Less and clear is better than more and unclear. Simple examples are better than complicated examples. Almost never I have felt or got the feedback that something was too simple. I am repeating in my head to not use the words “simply”, “just”, “easy”. If participants take home one or two points from a lesson, that’s for me success.

I prepare for the lesson by reading the instructor guide and all issues and open pull requests. I might not be able to solve issues, but I don't want to be surprised by known issues. I learn the material to a point where I know precisely what comes next and am never surprised by the next episode or slide. This allows me to skip and distill the essence and not read bullet point by bullet point.

I try to never deviate from the script and if I do, be very explicit about it.

A great exercise I can recommend is to watch a tutorial on a new programming language/tool you have never used. It can feel very overwhelming and fast to get all these new concepts and tools thrown at self. This can prepare me for how a participant might feel.

I find it very helpful if there is somebody else in the room who helps me detecting when I go too fast or become too confusing. I like when two instructors complement each other during a lesson but when doing that to others, I am often worried of interrupting their flow and timing too much.

A mistake I often do is to type too fast and in my mind I force myself to slow down.

Sabry Razick

My approach is to show it is fun to demystify concepts. Once a concept is not a mystery anymore, the learners will understand what it means, where it is coming from, why it is in place and what it could offer for their future. I try to relate concepts to real life with a twist of humour whenever possible if the outcome is certain not be offensive to any one. I use diagrams whenever possible, I have spent weeks creating diagrams that is sometime three or four sentences. That effort I consider worthwhile as my intention is not to teach, but to demystify. Once that is achieved, learners will learn the nitty gritty on their own easily and with confidence, when they have the use-case.

Juho Lehtonen

I'm gradually realising the different ways to get a hint whether the workshop participants are still following or perhaps bored. I assume it's communicating with the class, with exercises and simply by asking now and then. I also try to remember to observe how people look like (puzzled, bored) while I teach, not so obvious for me.

I believe that learners communicating with each other, in addition to with instructors and helpers, really helps them to understand things faster. (At least it helps me). So I try to make sure that no one sits or works alone at the workshops.

Richard Darst

Like many people, I've often been teaching, but rarely a teacher. I tend to teach like I am doing an informal mentorship. I've realized long ago that my most important lessons weren't learned in classes, but by a combination of seeing things done by friends and independent study after that. I've realized that teaching (the things I teach) is trying to correct these differences in backgrounds.

My main job is supporting computing infrastructure, so my teaching is very grounded in real-world problems. I'm often start at the very basics, because this is what I see missing most often.

When teaching, I like lots of audience questions and don't mind going off-script a bit (even though I know it should be minimized). I find that sufficient audience interest allows any lesson to be a success - you don't have to know everything perfectly, just show how you'd approach a problem.

João M. da Silva

I started giving technical trainings twenty years ago, and hence my perspective is perhaps more inclined towards the development of hands-on abilities and capability to solve problems, independently or in a team.

But the development of hands-on practical skills, requires some essential knowledge about the domain and some willingness to try different approaches in case one gets stuck. Some call this the "KSA approach" ("Knowledge-Skills-Attitude). Hence, I try to impart the essential knowledge (and where to find out more) at my trainings. And to encourage and challenge students in order to make them overcome their self-perceived limits (e.g. "I'm a Humanist, I can't use Python virtualenv").

I've been trying to study more about the Cognitive aspects of learning over the years, and I should find out the time to return to that. There's very interesting research in Problem Solving, with Learning being a important component in that domain.

Storytelling: humans are neurologically made for paying attention to good stories, and that's something that I try to put into account: to give a lesson like it would be a relevant narrative for the students, one that they could relate to and help them in their work

I like to draw and be creative with that, but have to pay attention to my handwriting during my trainings. I reckon that Architectural diagrams help students to understand the big picture, so I should invest more on those when development training material. I would also like to start looking into Concept Maps and Semantic Trees in training.

Interactive teaching style

What are the top issues new instructors face?

✓ Solution

- Breaks are not negotiable, minimum 10 minutes.
- Breakout sessions too short. Make them as long as possible, don't expect to come back for new intro, then go back.
- Get the speed correct. Not too fast and not too slow.
- People will accomplish less than you expect. Expect learners to be 5 times slower than you, at best!
- All the other tools and stuff will go wrong. Try to not bring in a dependency when you don't need it.
- Trying to accomplish too much: it's OK to cut out and adapt to the audience. Have a reserve session at the end you prepare, but are ready to skip.
- Explaining how, but not why.
- Running out of time to making your environment match the learner's.
- Running out of time to set up good screen sharing practices
- (terminal history, portion of screen, remote history) in advance.
- Assuming learners remember what they have already learned, or know the prerequisites. Or have stuff installed and configured.
- Not managing expectations: learners think that you will accomplish everything, and feel sad when you don't.
- Special issues when lessons delivered online (discussed during Workshop preparation and organization)

The Carpentries and CodeRefinery approaches to teaching

Here we will give you a very short overview of the Carpentries approach to teaching and highlight parts that are most important for teaching CodeRefinery style lessons.

Most CodeRefinery instructors have completed the [Carpentry instructor training workshop](#), which [anyone can apply for](#)

📌 This material

This section is derived from the [Carpentries instructor training material](#). We encourage you to further study this material later, and to sign up for a 2-day Carpentry instructor training workshop.

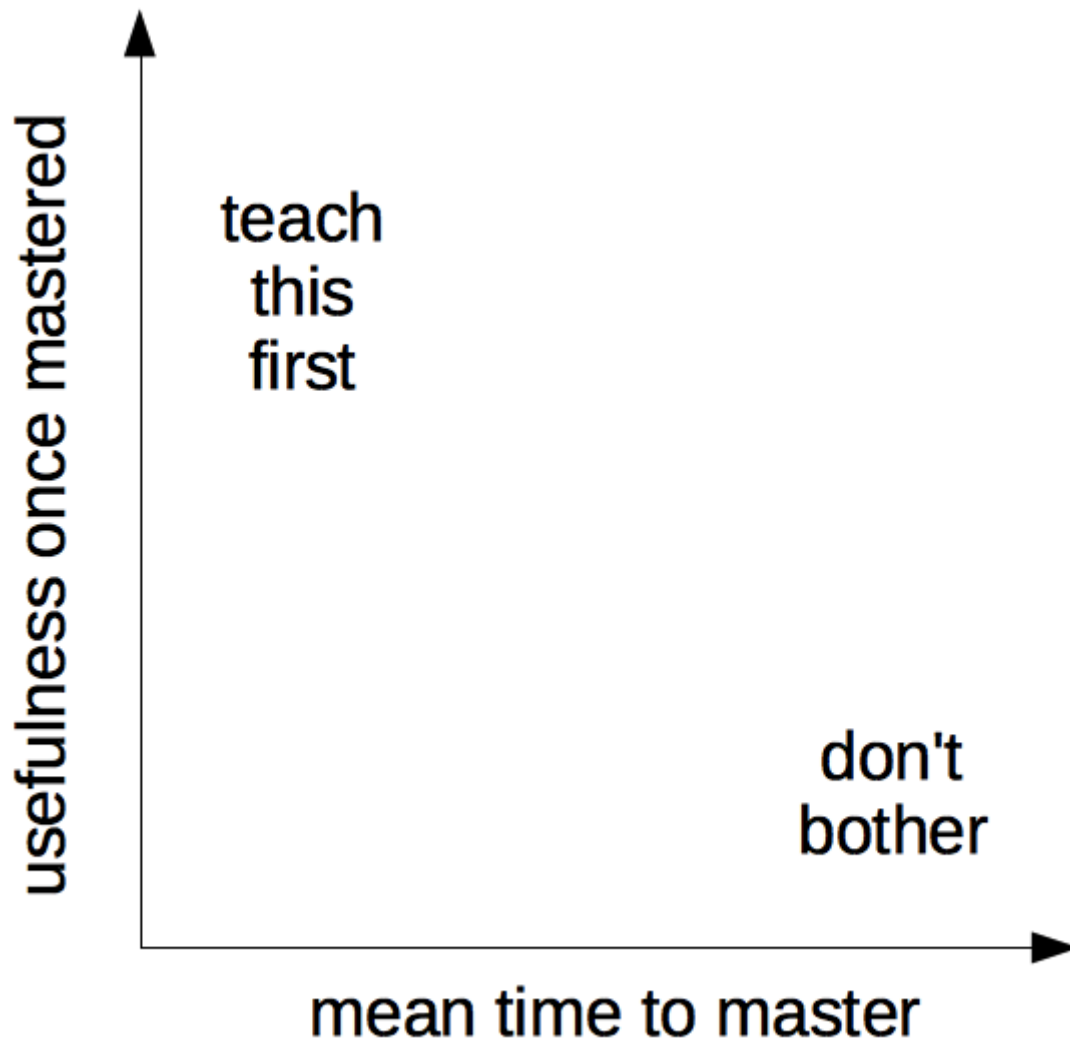
Key principles

The “Carpentries” approach to teaching is based on:

- Applying research-based teaching principles, especially as they apply to the Carpentries audience.
- Understanding the importance of a respectful and inclusive classroom environment.

Carpentries teaching principles

- Learners need to practice what they are learning in real time and get **feedback** on what they are doing. That is why the teaching approach relies on **live coding**.
- Learners best learn in a respectful classroom environment, so the Carpentries use a [Code of Conduct](#).
- Learners are encouraged to help each other during workshops as this improves their confidence and reinforces concepts taught.
- Carpentry instructors try to have learners do something that they think is useful in their daily work within **15 minutes of starting each lesson**.



In CodeRefinery, we follow The Carpentries teaching principles but in addition to **live coding** we often use **group discussions** to put in context the concepts we are teaching.

Applying these teaching principles are not sufficient and in addition we need to be able to check the effectiveness of our methods.

On the importance of feedback

Feedback is an essential part of effective learning. Feedback is bi-directional:

- To be effective, instructors need feedback on their learners' progress. Learners can also check their progress and ask relevant questions to get clarification.
- Instructors also need feedback on their teaching. For instance, this can help them to adapt the pace, add/skip optional exercises and improve their teaching.

Getting/giving feedback on learners' progress

This feedback comes through what is called *formative assessments* (in contrast to *summative assessment*).

❗ Summative Assessment

Summative assessment is used to judge whether a learner has reached an acceptable level of competence. Usually at the end of a course Learners either “pass” or “fail” a summative assessment. One example is a driving exam, which tells the rest of society whether someone can safely be allowed on the road. Most assessment done in university courses is summative, and is used to assign course grades.

❗ Formative assessment

Formative assessment takes place during teaching and learning. It sounds like a fancy term, but it can be used to describe any interaction or activity that provides feedback to both instructors and learners about learners' level of understanding of the material. For learners, this feedback can help focus their study efforts. For instructors, it allows them to refocus their instruction to respond to challenges that learners are facing. Used continuously

Learners don't “pass” or “fail” formative assessments; they are simply a feedback mechanism. For example, a music teacher might ask a learner to play a scale very slowly in order to see whether they are breathing correctly, and if not, what they should change.

Formative assessment is most useful when it happens frequently (we'll talk about how frequently later) and when the results are easily interpretable by the learner and instructor.

CodeRefinery uses different instruments to get feedback from learners:

- Surveys: we will discuss about CodeRefinery pre/post-surveys in the episode [Running a workshop: online](#).
- Exercises: we have many exercises during CodeRefinery workshops and use polls too but not necessarily many multiple-choice questions. This is something that we may change in the future but the initial reason was that we build on existing knowledge (see below section on our target audience) and give recommendations for best software practices:

there is no unique solution and you would like our learners to choose the approach that is most suitable for them. For the same reasons, we have many optional exercises to accommodate the different levels. We would like everyone to get something useful out of the CodeRefinery workshops.

Getting/giving feedback on teaching

Teaching is a skill. One of the objective of the CodeRefinery Instructor training is to give you the confidence in teaching CodeRefinery lessons. Later we will have group work where we will practice teaching some lessons.

Before doing so, we will learn here to give feedback on teaching using the same positive-vs-negative and content-vs-presentation rubric.

Give feedback on teaching (optional, 10 mn)

This exercise aims at learning to give feedback. It is optional as we have similar exercises when [practising teaching](#)). As a group, we will watch [this video of teaching](#) and give feedback on two axes: positive vs. negative and content vs. presentation. Have each person in the class add one point to a 2x2 grid on a whiteboard or in the shared notes (hackMD, etherpad, google doc) without duplicating any points. For online instructor training event, use breakout room (4-5 persons per group) to facilitate discussion. Then each group reports to the shared notes. You can use a [rubric](#) (used during The Carpentries teaching demos) to help you take notes. What did other people see that you missed? What did they think that you strongly agree or disagree with?

Who are the learners

The first task in teaching is to figure out who your learners are. The Carpentries approach is based on the work of researchers like [Patricia Benner](#), who applied the [Dreyfus model of skill acquisition](#) in her studies of [how nurses progress from novice to expert](#) (see also books by [Benner](#)). This work indicates that through practice and formal instruction, learners acquire skills and advance through distinct stages. In simplified form, the three stages of this model are:

Novices, competent practitioners and experts



- *Novice*: someone who doesn't know what they don't know, i.e., they don't yet know what the key ideas in the domain are or how they relate. One sign that someone is a novice is that their questions “aren't even wrong”.
 - Example: A *novice* learner in a Carpentries workshop might never have heard of the bash shell, and therefore may have no understanding of how it relates to their file system or other programs on their computer.
 - Example HPC: A learner who has never executed a program on remote computer in headless mode
 - Example HPC: A learner who has no understanding about using a queue system and having a hard time why a program can not be run directly after login in.
- *Competent practitioner*: someone who has enough understanding for everyday purposes. They won't know all the details of how something works and their understanding may not be entirely accurate, but it is sufficient for completing normal tasks with normal effort under normal circumstances.
 - Example: A *competent practitioner* in a Carpentries workshop might have used the shell before and understand how to move around directories and use individual programs, but they might not understand how they can fit these programs together to build scripts and automate large tasks.
 - Example: A *competent practitioner* in a CodeRefinery workshop is someone that understands the concepts of best software practices and its importance. He/she clearly sees the benefits of applying best software practices but he/she does not fully know yet how and what to use for their own projects.
 - Example HPC: Knows how to establish a connection to a cluster and have submitted jobs. But may not know how to request optimal amount of resources in a job and how to setup parallel jobs
- *Expert*: someone who can easily handle situations that are out of the ordinary.

- Example: An *expert* in a Carpentries workshop may have experience writing and running shell scripts and, when presented with a problem, immediately sees how these skills can be used to solve the problem.
- Example HPC: A learner who has a good understanding of the queue system, parallel processing and understand how to interpret error reports when something goes wrong and knows how to get help.

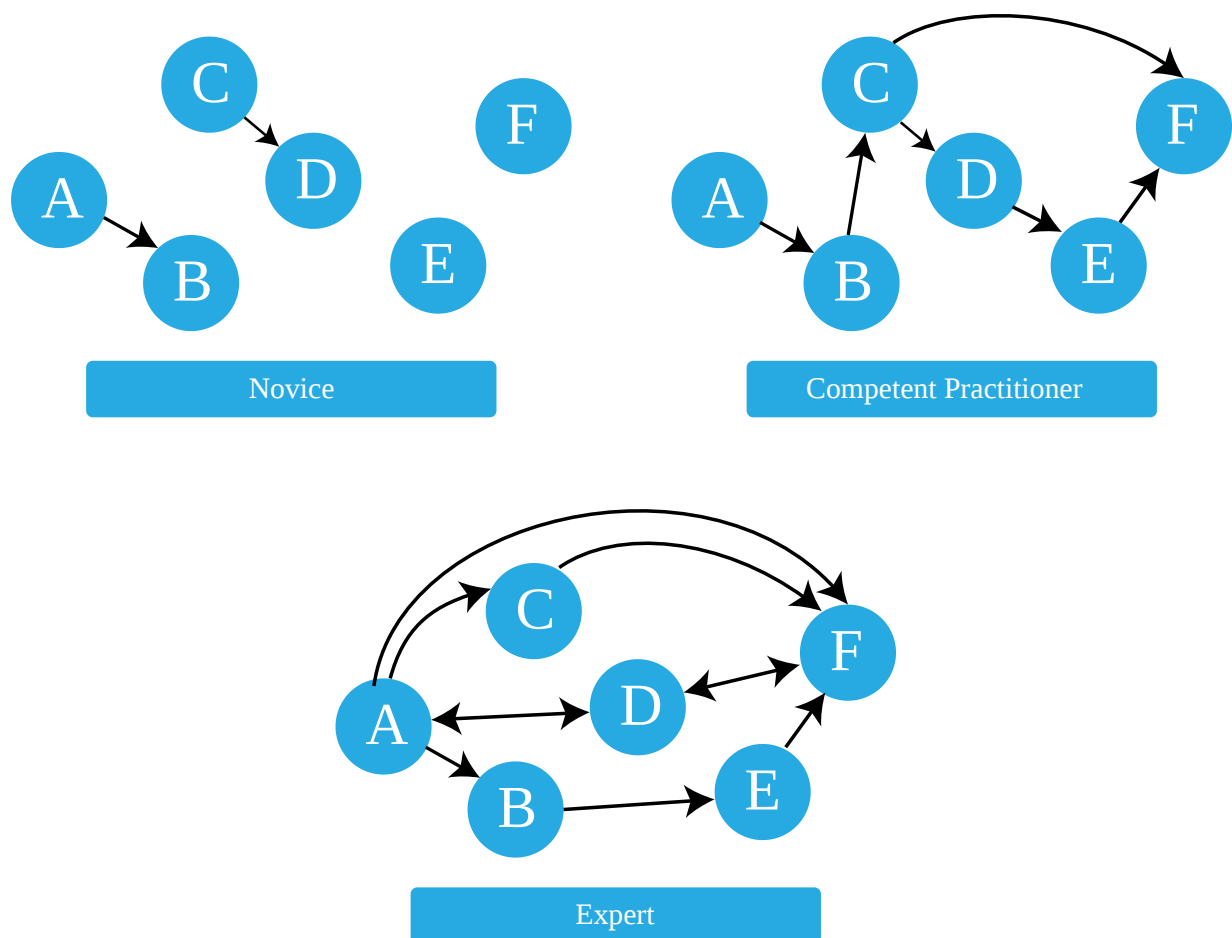
Cognitive Development and Mental Models

Effective learning is facilitated by the creation of a well-founded mental model. A mental model is a collection of concepts and facts, along with the relationships between those concepts, which a person has about a topic. For example, a long-time resident of the United States may have an advanced understanding of the location of US states, major cities and landmarks, weather patterns, regional economies and demographic patterns, as well as the relationships among these, compared with their understanding of these relationships for other countries. In other words, their mental model of the United States is more complex compared with their mental model of other countries.

We can distinguish between a *novice* and a *competent practitioner* for a given domain based on the complexity of their mental models.

- A *novice* is someone who has not yet built a mental model of the domain. They therefore reason by analogy and guesswork, borrowing bits and pieces of their mental models of other domains which seem superficially similar.
- A *competent practitioner* is someone who has a mental model that's good enough for everyday purposes. This model does not have to be completely accurate in order to be useful: for example, the average driver's mental model of how a car works probably doesn't include most of the complexities that a mechanical engineer would be concerned with.

We could expect a mixture of learners from *novice* and *competent practitioner* groups in HPC training events.



How “knowledge” gets in the way

Mental models are hardly ever built from scratch. Every learner comes to a topic with some amount of information, ideas and opinions about the topic. This is true even in the case where a learner can’t articulate their prior knowledge and beliefs.

In many cases, this prior knowledge is incomplete or inaccurate. Inaccurate beliefs can be termed “misconceptions” and can impede learning by making it more difficult for learners to incorporate new, correct information into their mental models. Correcting learners’ misconceptions is at least as important as presenting them with correct information. Broadly speaking, misconceptions fall into three categories:

- **Simple factual errors**, such as believing that Vancouver is the capital of British Columbia. These are the easiest to correct.
- **Broken models**, such as believing that motion and acceleration must be in the same direction. We can address these by having learners reason through examples to see contradictions.
- **Fundamental beliefs**, such as “the world is only a few thousand years old” or “human beings cannot affect the planet’s climate”. These beliefs are deeply connected to the learner’s social identity and are the hardest to change.

The current HPC carpentry workshop material are aimed at **Novice** of HPC

Among *Novice* learners there might be learners who are experts in their domain and very competent in the program they are executing, but may not have used a HPC system before. Then among the **competent practitioners** There might be learners who repeat some procedures they have inherited but lack a in-depth understanding of what's going on. That is why it is important to get accurate feedback, before and during the workshops to understand the learner profiles.

Exercise: How to identify learner profiles?

1. How to identify learner profiles from surveys and during the class
2. Which types of learners should the lessons focus on

CodeRefinery Curriculum and Reverse Instructional Design (with recommendations for HPC carpentries)

When writing a CodeRefinery lesson, we take a “reverse” approach to instruction, as described in Wiggins and McTighe’s [Understanding by Design](#), that keeps the focus firmly on learning outcomes. The order of preparation in this case becomes

- Determine your learning objectives
- Decide what constitutes evidence that objectives have been met, and design assessments to target that evidence
- Design instruction: Sort assessments in order of increasing complexity, and write content that connects everything together

Working with learning objectives

Each CodeRefinery lesson (also the HPC carpentries lessons) usually has a *learning objectives* section. Good learning objectives are quite specific about the intended effect of a lesson on its learners. We aim to create learning objectives that are specific, accurate, and informative for both learners and instructors.

Using Bloom’s Taxonomy to write effective learning objectives

[Bloom’s Taxonomy](#) is a framework for thinking about learning that breaks progress down into discrete, hierarchical steps. While many ideas have come and gone in education, Bloom’s has remained a useful tool for educators, in particular because the hierarchy seems to be reasonably valid: outcomes at the top of the hierarchy cannot be achieved without mastery of outcomes at the bottom. In the long term, everybody wants to be at the top. However, in aiming to meet learners where they are, we also need to be mindful about helping them to “[grow a level](#),” helping them to recognize when they have achieved that growth, and guiding them to look ahead to where we might not be able to take them.

Bloom's Taxonomy

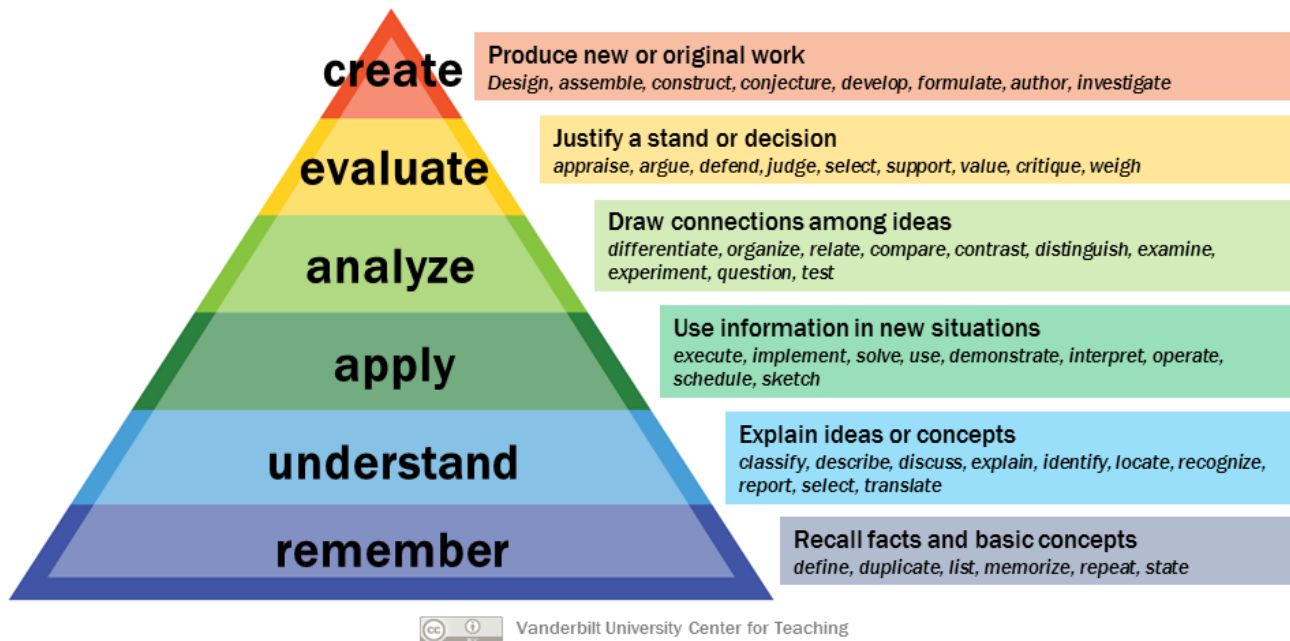


Image credit: Vanderbilt University Center for Teaching

Revisiting Learning objectives

When using existing teaching material, *reverse instructional design* principles might be applied as follows:

1. Review the lesson's learning objectives carefully, thinking about how they will work for your audience
2. Scan the lesson to identify promising points to check in with your learners, using formative assessment to verify that objectives have been met
3. Review the connecting content in detail to be sure everything works and you have anticipated likely problems and questions.

We strongly encourage you to read them before teaching a lesson and to review whether they still match the content of the lesson:

! Workshop manuals

CodeRefinery maintains a number of [workshop manuals](#) with most of the "primary" information. This episode condenses this into a quick overview.

Running a workshop: online

Online teaching discussion

! Discussion: Online vs in-person

In notes:

- Compare and contrast the benefits of online teaching with in-person: {advantage, disadvantage} × {content, presentation}
- How do you have to prepare differently?
- What are your own experiences?

Case study: Mega-CodeRefinery and Finland HPC Kickstart

- Mega-CodeRefinery
 - Audience of around 90-100
 - “bring your own breakout room” (see below)
 - 3 days/week, 6 days total
 - Lessons as normal in CodeRefinery
- HPC Kickstart
 - 250 registered, ~180 max participants
 - Multi-university: local differences made this much harder to manage.
 - Breakout rooms not pre-planned.

Mega-CodeRefinery worked very well, HPC kickstart didn't - but not because of the size.

General workshop arrangements

! Manuals link

- [before the workshop](#)

- Select a coordinator, recruit instructors (at least 3 is important), find helpers
- Find a good lecture room: [requirements](#)
- Set up workshop webpage using the [Github, template repository] (<https://github.com/coderefinery/template-workshop-webpage>): [see manuals](#)
- Advertising the workshop
- Communication with registered participants

CodeRefinery online scaling strategy

- We started online workshops in 2020 March, for the obvious reasons.
- First, we started with two “normal size” (20 people) practice workshops
- Then we did a 100 person workshop. It went well, but there is less tolerance for problems.

Basic preparation

- You need more breaks are needed
- People have a way of doing too many things and not focusing.
- “[How to attend an online workshop](#)” guide to prepare learners

Basic platform: Zoom

- Zoom (not the most ethical, but worked well and was available)
- [Zoom mechanics: instructions for students](#).
 - Mostly things that are known
 - We don't use Zoom interaction features much anymore (faster/slower/etc), but breakout rooms and HackMD instead
- See also: [Online training manual](#) (which is getting a bit old compared to what is below).

Breakout rooms, bring your own team

- Breakout rooms are
 - Static: same people across whole workshop
 - Contain one helper per room (see below)
- Team registration: accept a “team” field when registering, people on the same team are put together.
 - Gives motivations for learners to bring their colleagues and learn together.
 - More than one person learning together greatly increases update
- You need a powerful enough registration system to assign rooms and email them to people!
- We ask people to name themselves “(N) Firstname Lastname” or “(N,H) Firstname Lastname” for helpers. Then it is fast to assign them to their designated breakout rooms.
- See also: [Breakout room mechanics](#)

Helper training

- Each breakout room has a helper
- Helper should be a little bit familiar, but not expected to be able to answer all questions.
- Special, custom [helper training](#) since helpers make or break the workshop
- Helper recruitment:
 - Our networks
 - Team registration: if a team registers with their own helper, then they are guaranteed to get in together. “bring your own breakout room”
 - Former learners, ask them to come back.
- Two helper trainings the week before the workshop.

Staff roles

To reduce stress on any one person, we clearly define the different roles and try to avoid overlap. We actually have enough people for all of these, so it works well.

- Workshop coordinator
 - Registration, etc.
- Zoom host
 - Handles registration, breakout rooms, recording, Zoom chat.

- HackMD helper
 - Dedicated to watching HackMD and answering questions quickly.
 - [Host on manuals](#)
- Expert helpers
 - “Spare hands” who rotate between breakout rooms and make sure helpers are doing well.
 - Give feedback to instructor about how breakout rooms are going.
 - Take the place of missing helpers.
 - Easy way for any people with a bit of spare time to help out.
 - [Expert helpers in workshop](#)
- Instructors
 - Teach, they shouldn't overlap with the above roles (but serve as expert helpers other times).
 - Usually also improve the lesson a bit before teaching
 - [General staff intro in manuals](#)
- Workshop preparation meeting
 - Get together, introduce roles, kickstart instructors
 - [Workshop prep meeting in manuals](#)

HackMD

- We've been using it here
- Chat doesn't work when large, written document does.
- HackMD can just about scale to ~100 person workshop. Recommend learners keep it in view mode while not editing.
- Voice questions are still allowed, but will be recorded. Staff raise important questions from HackMD to the instructor immediately.
- HackMD also allows communication when in breakout rooms.
- You can get multiple answers, and answers can be improved over time.
- [HackMD mechanics](#) and [HackMD helpers](#).

Recording and streaming

- When you have 100 people, main room is quiet anyway: you don't lose much by recording.
 - Questions anonymously in HackMD, privacy loss is not so bad
- Breakout rooms are never recorded
- Streaming
 - We streamed via Twitch: <https://twitch.tv/coderefinery>
 - We typically get 5-40 viewers.
 - Zoom can directly send the stream to Twitch: no extra software needed.
 - Twitch archives videos for 14 days, which allows learners to get an instant reply (we get hundreds of views in the next days).
 - So while possibly not useful for new people to learn, the instant reply is very useful. Instructor can also work on problems in main stream during breakout rooms, which learners can watch later.

- Streamers also have access to HackMD to ask questions.
- Certain tricks needed to keep learners from appearing in recording or stream
 - “Spotlight video”, host does not go to gallery view, uses dual monitor mode. We are still figuring this out.

Installation time

- People *have* to be ready once we start, or else everything fails.
- Two installation help times the week before.
- Every email emphasizes that you have to be prepared, and “requires” you to attend workshops (but really it’s only)
- Installation instructions include *steps to verify*
- Installation instructions also include *video demonstrations* of installation and verification.
- We haven’t had that many installation problems, but also we keep the requirements simple.
- Helper introduction is right before software install time, so helpers can stay and help with install if they want.
- *Design to be easy to install and get set up.*

Other notes

- Make breakout sessions as long as possible: 10 minutes is really too short. 20 minutes is a good minimum time.
- Be very clear about exercise expectations
- Keep HackMD updated as a log.
- Don’t combine breaks and breakout times.
- The more people you have, the more diverse audience you have and the more people overwhelmed and underwhelmed.

Workshop collaborations

Why limit ourselves to CodeRefinery workshop? Why not use our network and techniques for more

- Case study: [Python for Scientific Computing](#)
 - Started by Aalto
 - Announced to CodeRefinery, five more instructors from three countries joined.
 - Rapid collaboration, taught course shortly later.
 - Announced to all institutions. Some places had physical rooms, some were pure online
 - Also streamed
 - It was much more fun and less stressful to work together
- We want to continue this kind of collaboration in other workshops.

How to teach online

📌 Objectives

- Understand the benefits and disadvantage of online teaching, compared to in-person

- Set up a good screen share
- Understand the benefits and disadvantages of team teaching
- Prepare for the teaching practice

Why teaching mechanics matter

- When you teach, you are mainly showing a basic example for the learner to follow along
- The learner has a *lot* more to think about than you do, so you need to minimize the possible distractions and unnecessary weirdness.
- A learner will often only one small screen, limiting the number of things that they can think about.
- You are must faster than learners (5 times possibly?) You have to do things to slow yourself down.
- It's easy to save these mechanics until the end, and then you run out of time.

Shell sharing

Discussion: what goes into a good shell share or demonstration?

When you are following along with a type-along demonstration, what things:

- Are useful to make it easy to follow along
- Make it harder to follow along

Answer in the collaborative notes

When doing any demonstration, there are difficulties:

- If one misses something, you can't rewind to see it - is there any way to catch up?
- The learner must get oriented with the whole picture, while instructor knows precisely where to focus.

A good **shell share** has some of the following properties:

- Large font
- Shell history, available separately from the main shell window
- Closely matches the type-along instructions

We have a collection of shell sharing systems:

- We will look over [lesson presentation hints](#).
- There are other things you can copy
- Whatever you do, do *something*.

Discussion

The instructor will demonstrate several shell-sharing systems. You will use this in the teaching practice.

Screen sharing

Discussion

Look at the various [screen layouts in the CodeRefinery manuals](#). Use the HackMD to comment about what which are better or worse.

📌 HackMD prototype

```
- S1
  - good:
  - bad:
- S2
  - good:
  - bad:
- S3
  - good:
  - bad:
- S4
  - good:
  - bad:
- Student layouts:
  - ...
- Instructor layouts:
  - ...
```

- Many learners will have a smaller screen than you.
- You should plan for learners with only one small screen.
- A learner will **need to focus on both your screen share and their work**.
- Sharing your a whole screen is almost always a bad idea, if you want the learners to do anything at the same time.
- If you constrict yourself, then your experience is more similar to that of a learner.

Vertical sharing:

- CodeRefinery has recently started trialing a **vertical share** system, where you share a vertical half of your screen.
- This allows learners with one screen to display your screen side-by-side with their learn
- Zoom provides a “Share a part of screen” that is good for this.

Meta-talk

Don't just teach, also make sure you guide the learners through the course.

- You know what you just discussed, and what is coming next, but learners are often stuck thinking about now.
- Give a lot of “meta-talk” that is not just about the topic you are teaching, but how you are teaching it.
- Examples
 - **Why** you are doing each episode
 - What is the purpose of each exercise
 - Clearly state what someone should accomplish in each exercise and how long it will take - don't assume this is obvious.
 - What is the point of each lesson. How much should people expect to get from it? Should you follow everything, or are some things advanced and optional? Make that clear.

Teach teaching

- Demonstration-based teaching require two different types of focus:
 - Doing the mechanical steps as a demonstration
 - Explaining why you are doing it
- This is a lot for one person to keep in mind, so can multiple people work together for this?
- Team teaching idea:
 - One person is doing the demonstrations
 - One person is giving the commentary about what they are doing
 - The lecture becomes a discussion between two people instead.

Advantages:

- This reduces the pressure on each person (reduces demo effect)
- You are less likely to forget things
- It slows you down in teaching
- It makes the lesson more interesting to listen to
- One person can follow questions
- Great for introducing new instructors (which half is easier to start with?)

Disadvantage:

- Requires two people's time
- Requires coordination when preparing (slows you down in preparation)
- Unfamiliar concept to most people

Questions

- Questions are great, and important for any practical and interactive class
- But questions in main room doesn't scale to very large rooms.
- CodeRefinery strategy: HackMD for questions
 - Chat is not good enough, you can't reply to old things
 - HackMD allows threaded replies and follow up later

- You need some other helpers to watch HackMD and answer, and bring things up to you. And let you know how things are going.
- Learners can ask anonymously
- Learners don't have to worry about interrupting the flow.
- Disadvantage: can produce information overload, warn people to not follow too closely
- With too few people, it can turn out to be very quiet.
- We will learn more about HackMD questions tomorrow in [Running a workshop: online](#).

→ See also

- [Running a workshop: online](#)
- [HackMD mechanics](#) and [HackMD helpers](#) on CodeRefinery manulas.

Teaching practice

In [Teaching practice and feedback](#), you will break into groups and try to apply these strategies to a five-minute example session.

See also

In this lesson:

- [Running a workshop: online](#)

CodeRefinery manuals:

- [Instructor tech setup](#)
- [Lesson preparation hints](#) (more focused on in-person)
- [Instructor introduction](#) - has a lot of tips for new instructors, but also more things about the workshop.
- [Workshop prep call](#)

Teaching practice and feedback

Goals of the teaching practice:

- In groups of 4-5 persons we will practice teaching a **5-minute segment of a lesson of your choice**.
- The section you pick should require **screen sharing** and be of some **demonstration or follow-along task** (preferably using a shell) to also practice having a good screen-sharing setup.
- We will practice giving **constructive feedback**.
- We will practice improving our 5-minute segment by taking the feedback into account.
- In both session you can teach the same topic/segment but if you prefer you can also change the topic/aspect for the second session.

Instructor demo

- In order to demonstrate the goals of this section, the instructor will make a 5-minute demo for your evaluation.
 - It is designed to include some good and bad practices for you to notice.
-

Teaching demos part 1

In group rooms, 50 minutes.



Exercise

- We organize the breakout rooms to not only discuss one lesson/topic so that it is more interesting to listen and also probably we will all get more useful feedback.
- Give each other **constructive verbal feedback** on the teaching demos, for example using [this demo rubric](#).
- Write down questions (in the collaborative document) that you would like to discuss in the main room or interesting conclusions which you would like to share with others.

Teaching demos, part 2

In group rooms, 50 minutes.



Exercise

- In the second round we distribute the rooms differently so that you can present it to a **new group of workshop participants** and can receive new feedback.
- Ask for feedback and one/few point(s) you want to improve.
- In your second trial check whether you feel the demonstration improved.
- Share your lessons learned in the collaborative document.
- Give us also feedback on this exercise format. Was it useful? What can we improve?

Discussion



Main room discussion

- We discuss questions and conclusions which came up during the group work session.
-

Optional: feedback for two live-coding examples

Exercise

Teaching by live coding is a [performance art which requires practice](#). This exercise highlights some typical pitfalls that most instructors fall into sooner or later, and also shows how to avoid them. Watch closely since we will be giving feedback!

- Watch these two videos: [video 1](#) and [video 2](#)
- What was better in video 1 and what was better in video 2?
- Please give feedback in the shared workshop document.