CodeRefinery manuals

The CodeRefinery team

Jun 30, 2020

LEARNERS

1 Zoom mechanics and controls 3

2 HackMD mechanics and controls 5

3 Tips for helpers 7

4 Hints on breakout rooms, teams, and helping 9

5 Lesson presentation hints 13

6 Summary of Teaching Tech Together 17

7 Workshop requirements - in person 23

8 Organizing a CodeRefinery workshop 25

9 Online training manual 31

10 Guide on how to use Indico for managing workshops 35

11 Icebreakers 39

12 Lesson design 41

13 Lesson review 47

14 Writing technical docs 49

This site contains various manuals about CodeRefinery workshops and teaching/lesson development in general.

CHAPTER

ONE

ZOOM MECHANICS AND CONTROLS

- *How to mute and unmute*
- *Please use your real name (instead of a system default username)*
- *Indicate in your name if you are in a team and/or if you are a helper*
- *How to signal if you are away from keyboard*
- *How to signal when you completed a task successfully*
- *How to ask a question*
- *How to signal a technical problem or that you got stuck*
- *How to give feedback on the speed*

1.1 How to mute and unmute

In lower left corner of the client you can mute and unmute yourself.

In the *main room* during lectures, it is best to keep your microphone muted in the main room unless you want to say something. It's OK to unmute and speak up.

If you are in a quiet place, it's best to stay unmuted in breakout rooms and during discussions. This will make discussion much smoother - a quiet environment or headset microphone helps with the flow a lot.

1.2 Please use your real name (instead of a system default username)

Click on "Participants" (bottom, middle):

You can rename yourself by clicking the blue "Rename" next to your name:

1.3 Indicate in your name if you are in a team and/or if you are a helper

If you are part of a team, please indicate your team name or number like this:

If you are a helper, please indicate also:

This makes it easier for the workshop organizers to manage breakout rooms.

1.4 How to signal if you are away from keyboard

Please select the "clock" symbol if you are away or otherwise busy:

1.5 How to signal when you completed a task successfully

We will sometimes ask you to signal to us once you have successfully completed an exercise or type-along step. You can do this using the green check symbol:

1.6 How to ask a question

If you want to ask a question please use the "hand" symbol:

If this symbol is not present in your Zoom client, you can type "\hand" in the chat window:

1.7 How to signal a technical problem or that you got stuck

If you hit a technical problem or got stuck somewhere in an exercise or type-along, please let us know with the red circle symbol:

We will then probably ask you to unmute and briefly describe the problem and then based on the problem and timing we may assign you into a separate virtual room with a helper where they can resolve the problem.

Once we have assigned you a helper we will ask you and the helper to "Join Breakout Room" (bottom right):

1.8 How to give feedback on the speed

There are also signals for faster and slower and with this you can indicate to us whether we should adjust the speed.

1.9 Zoom doesn't have to fullscreen when someone shares their screen.

By default, when someone shares their screen, Zoom goes into fullscreen mode. This can be inconvenient when you need to see multiple windows at once. You can disable this with Settings → Screen Share → "Enter full screen when participants share".

CHAPTER

TWO

HACKMD MECHANICS AND CONTROLS

Hackmd is a syncing text editor online. We use it to answer questions and provide feedback without interrupting the main flow of the room. Also, everyone can have a more equal share of time.

2.1 Privacy

- Assume the HackMD is public and published: you never need to put your name there.
- The HackMD will be published on the website afterwards. We will remove all non-instructors names, but it's easier if you don't add it at the start.
- Please keep the link private during the workshop, since since security is "editable by those who have the link".
- You can use `[name=YOURNAME]`, to name yourself. We *will* leave these in unless they are negative comments.

2.2 Basic controls

- At the top (left or right), you can switch between **view**, **edit**, and **split view and edit** modes.
- You write in markdown here. Don't worry about the syntax, just see what others do and try to be like that! Someone will come and fix any problems there may be.
- Please go back to view mode if you think you won't edit for a while - it will still live update.

2.3 Asking questions

Always ask questions and add new sections at the very bottom (unless you are answering to something old, but we don't look up that much). You can also answer and comment on other questions, too.

Other hints:

- Please leave some blank lines at the bottom
- Please don't "select all", it highlights for everyone and adds a risk of losing data (there are periodic backups, but not instant).
- It can be quite demanding to follow the HackMD closely. Keep an eye on it, but consider how distracted you may get from the course. For things beyond the scope of the course, we may come back and answer later.

CHAPTER

THREE

TIPS FOR HELPERS

See also: *breakout-rooms-helping.md* for information on how to manage online breakout rooms.

3.1 Preparing to help

As a helper, we do not expect you to know all our CodeRefinery training material but if you have time:

- Read through the instructor guides for the lessons (ask instructors if you do not know where to find them).
- Be ready to introduce yourself in one or two sentences: think about what you would like to convey as helpers to the classroom. How did (or does) CodeRefinery help you?

3.2 Tips for helping during the workshop

3.2.1 Code of Conduct

Teaching isn't about helping those who are already "in", it is for those who aren't. Thus, we follow The Carpentries Code of Conduct for all our interactions before, during and after workshops.

If you see anything that is not supporting an equal learning environment, please mention it to one of instructors.

3.2.2 Creating a positive learning environment

As a helper, you have a crucial role during workshops:

- Encourage learners to learn from each other.
- Acknowledge that some of the material can be difficult and that they will learn more working together.
- Acknowledge when learners are confused and raise it to the instructors. Understanding why learners are confused provides useful feedback for instructors. You should be our eyes and ears.
- As we said, you don't have to know everything, just like learners don't necessarily know everything (we don't know everything, either). It's more important to be responsive and work together.
- Stand up and walk around, try to make rounds by everyone. If you are convenient, students will ask. If you are sitting in the back, student's wont. Students rarely try to get your attention from across the room if you don't look ready.

3.2.3 Things you should not do in a workshop

- Take over the learner's keyboard. It is rarely a good idea to type anything for your learners and it can be demotivating for the learner because it implies you don't think they can do it themselves or that you don't want to wait for them. It also wastes a valuable opportunity for them to develop muscle memory and other skills that are essential for independent work. Instead, try to have a sticky note pad and pen and write the commands that they should type.
- Criticize certain programs, operating systems, or GUI applications, or learners who use them. (Excel, Windows, etc.)
- Talk contemptuously or with scorn about any tool. Regardless of its shortcomings, many of your learners may be using that tool. Convincing someone to change their practices is much harder when they think you disdain them.
- Dive into complex or detailed technical discussion with the one or two people in the audience who have advanced knowledge and may not actually need to be at the workshop.
- Pretend to know more than you do. People will actually trust you more if you are frank about the limitations of your knowledge, and will be more likely to ask questions and seek help.
- Use "just", "easy", or other demotivating words. These signal to the learner that the instructor thinks their problem is trivial and by extension that they therefore must be stupid for not being able to figure it out.
- Feign surprise at learners not knowing something. Saying things like "I can't believe you don't know X" or "You've never heard of Y?" signals to the learner that they do not have some required pre-knowledge of the material you are teaching, that they don't belong at the workshop, and it may prevent them from asking questions in the future.

3.3 From helper to instructor

You have been a helper at a CodeRefinery workshop and it was a great experience for you so you are willing to take an active part to CodeRefinery?

We are welcoming everybody to the CodeRefinery Community and strongly encourage and support you if you are willing to teach CodeRefinery workshop.

We use Zulip to discuss within our team and community. We discuss in the open and you can join us on https://coderefinery.zulipchat.com: you can listen in, follow certain threads, participate, and influence.

3.3.1 What knowledge is needed?

If you have been helping out at a CodeRefinery workshop, you are most likely familiar with our lessons.

- Are there any lessons you would prefer to teach?
- Do you need any inputs from CodeRefinery to get the confidence to teach a CodeRefinery workshop?

Do not hesitate to contact us and let us know you are willing to teach CodeRefinery workshops!

CHAPTER

FOUR

HINTS ON BREAKOUT ROOMS, TEAMS, AND HELPING

This page is more targeted to online workshops, but could be relevant to in-person too.

As a helper, you are what can make a breakout room / group work very good or just normal. This is a lot of responsibility and isn't easy, but it's also not that complex: you aren't expected to know everything, but instead focus on the flow. Our guidelines below should make it doable for everyone.

Here, we give some hints on making the most of breakout rooms. This is especially targeted towards helpers.

- A **helper** has basic knowledge and keeps one breakout room going.
- A **expert helper/instructor** is someone who knows material well, possibly an instructor. They tend to move around helping different groups.

4.1 Background: hierarchical workshops to scale

Traditionally, a workshops has instructors and helpers, but the capacity is limited by instructors, so we are limited to ~30-40 people at most. Then, we tried to scale to larger numbers: even up to 100 people. For this, we have to rely on *helpers* a lot more, to run a breakout room. A helper does not have to be an expert in the material, but should be able to keep things flowing.

4.2 Teams

We try to arrange people in teams which stay together for all breakout sessions on all days. This allows people to form a bond and get the rooms started sooner. We will try to keep you in the same breakout room as long as we can, but we give no promises and will rearrange as needed when people can't attend.

We sometimes allow people to register as teams: you bring learners and a helper together, with the hope that you can keep working together afterwards.

4.3 General, in the main session

As an instructor, you first need to decide how to balance between the main room and breakout sessions.

- **Clearly say when a learner watches, when they type along, when they should work on something independently as an exercise.**
- You can try to plan the lesson so that more of it gets pushed to the breakout session, and the main room is mostly discussion. Since each room has a helper, this works a bit better than in in-person workshops
  - This doesn't work so well for all lessons. Especially the more basic lessons don't work so well

  - It also requires some care, and you should probably hop from room to room to monitor the progress.

4.4 Preparing for the breakouts (in the main room)

As an instructor, you need to clearly define what the tasks of each breakout session is (even if it is just "explore and discuss"). Online courses need more **"meta talk"** about how you expect things to go, since it's not as easy to read the room or fill in expectations later (distractions, hard to communicate to breakout rooms after opened).

- Clearly say what the tasks of the breakout session will be.
- Put that task and a link to the part of the lesson in the hackmd.
- Clearly say how long each breakout session will be (make sure it's long enough)
- Try to make breakout sessions longer:
  - imagine a 5 minute overhead for each session, getting people there, deciding who does what, acquainted with what they need to do, and debugging problems.
  - 10 minutes is quite short, 20 minutes is best.
  - Can you say less and let people discover it for themselves?

As a helper, if anything is unclear to you, it is very unclear to others. Speak up and ask!

4.5 Helpers in breakout rooms

As a breakout room helper, your main task is to *keep people talking and interacting, understand their difficulties, and encourage them to work on the exercises together.*

- You can always start by greeting people and asking how the lesson is going
- You can use chat within breakout rooms: Chat to "Everyone" in a breakout room only means people in that room.
- There are several strategies below. Combine as needed - read the room and see what they want, but do provide encouragement to do something.
- If you need help, there is "Ask for help" in the meeting controls. Click that, and the host will send someone. You can also write the request (with more details) on the hackmd, someone should be watching it and relay it to the host.
- Watch the time and try to keep things moving. If some debugging takes to long, it's reasonable to ask for another helper who might have seen the problem.
  - If any one problem takes too long, it's OK to say "we don't have time, let's come back"
  - Or, ask for an expert helper to come by and maybe answer quickly, or break off to work on a solution.

There are several strategies you can use to run your breakout room:

Strategy 1:

- **Helpers ask someone to share the screen and go through the exercise.**
- You can encourage the others to guide the one who is sharing the screen. Or let the person go on her/his own pace.
  - I joke "The person who shares the screen has the easiest job, since the rest of us guide you."

        * This might go against the CodeRefinery's code of conduct. The wording and the joking might cause some people feel inferior. So, I recommend not to do so! Please others comment about what they think about this.

- Try to alternate who is sharing the screen for each session.
- When someone has an issue, of course you switch screen share to them and maybe even continue from there

Strategy 2:

- **Everyone does the exercises themselves, and once someone has a question, encourage them to share the screen and you discuss.**
- If everyone is active, this can be good, but there is a risk that no one starts off.

Strategy 3:

- **You can also share the screen if no one is willing to.**
- It might be good to give learners some lead first, and use this only if no one volunteers.

4.5.1 Common problems

- One learner asks very many questions, ends up monopolizing all of the time. Other learners are left without help, and the whole group may not get the exercises done
  - Call an expert helper to the room. They should be circulating, so let them know to spend some more time
  - It can be very hard to say "no", but it's more important to have balance than answer every question you are asked. If you need to say no, you can try things such as "I'm sorry, but in order to finish we need to go on now. We can keep working on it later - would you like to watch?"
- There is some sort of problem that ends up taking a lot of time
  - Work on it for a minute or two.
  - Ask for an expert helper to drop by, by writing in the HackMD. Nothing wrong with this.

4.6 Expert helpers

There's not much difference between a helper and expert helper, but we envision this role standing by and jumping into rooms when there's a difficult problem.

- Sometimes, you wait around for a problem that needs your attention. But it's better to be proactive and go into the rooms yourself and check them out. Talk to the organizers/instructors to see which you should do.
- You aren't assigned to particular breakout room, but **you can switch between them** (but it's not obvious how):
  - To do this, you *do* get assigned into one room initially. Join that room. *After* you are in the room, click on "Breakout Rooms", and then `Join` to switch to a different room of your choice.
  - You also always have the option "Leave breakout room" (if in a room) or "Join your assigned room" (if in main room and assigned one).
- Your role is to switch between breakout rooms and check up on them.
  - e.g. join room 1, take a look/ask how it is, then join room 2, then 3, then back to 1, and repeat.
  - Of course, stay in one longer, if it's needed.
  - Make a note of any important questions to be asked in the main room afterwards.

- Try to divide up the breakout rooms between the staff, and try to join and catch up with the same rooms (this promotes familiarity).
  - E.g. A rotates between rooms 1-3, B gets rooms 4-7, C gets rooms 8-11.
- Make sure to watch the HackMD for helper requests, this could help you decide which room to jump to next. Comment when you are heading there.

Concrete example for an expert helper's time:

- I join breakout room 5 randomly. I spend 15 seconds watching, then ask if things are going OK. If everything is good, I move on within a minute since I am not needed (if there is a good break, I'll ask "everything OK? good, see you around."). If there are questions that I can help with, I answer them. If they seem to be struggling, then I will make a note in the HackMD and stay a while longer and watch/help.

4.7 Common issues and solutions

- A room is very slow, the person sharing the screen is working quite slowly.
  - Kindly suggest that you or someone else take over and go through it faster
  - Yes, this is hard to say nicely
- No one wants to take initiative and screen share
  - If you think everyone is confident enough, this can be OK
  - But especially at the beginning of the workshops, you can share your own screen and go along with people.
- Someone is having trouble installing software
  - "Perhaps we can take a look at this after the workshop? We try to make sure everything is installed beforehand, but "

CHAPTER

FIVE

LESSON PRESENTATION HINTS

This is a checklist/hints on what to do when standing up and giving a presentation.

5.1 Before each lesson

5.1.1 Non-technical

- Remember: sticky notes, water, extra whiteboard markers.
- Make your text large enough to be seen in the back, then bigger. Make your voice loud enough to be heard in the back, then louder.
- As people are coming in, encourage them to sit next to someone with a similar operating system - then, when helping each other, the unimportant differences are minimized.
- By the same token, don't allow people to sit alone: ask everyone to set next to at least one other person. That way, people can help each other.
- Have a pen and paper next to you. When you notice problems in the material, write it down right away during breaks in the type-along parts.
- Set up feedback system (chat, questions, etc)

5.1.2 Technical

- Move your `.bashrc` and `.gitconfig` files to somewhere else before you begin. You want your environment to match the student's as closely as possible. Consider setting `export PS1="\w $ "` in terminal.
- Create a nice, large shell window with good contrast on the screen. Beware of colorized text, such as the red in "git diff".
- Make sure the text is more than large enough - people are not just reading, but struggling to find the important parts.

5.2 Screencasting

Set an easily-viewable prompt. Colors may be good, or if not have a newline (don't little minimal color and no spacing between commands, it is hard to parse what's a command and what's an output.) The minimum is `export PS1='\n\w \$ '`. With color is `export PS1='\n\[\e[0;36m\]\w \$\[\e[0m\] '`.

If you are doing live shell work, you will have commands and output all interleaved, which makes it hard to follow what you actually typed. Have a separate window that shows recent commands only, without output. Arrange your screen so there is the main window and the smaller "history" window. The history window runs the tail commands and can be used as a reference for what you just did.

Also check the shell exporter by sabryr, which copies recent history to a remote server.

Simple: The simple way is `PROMPT_COMMAND="history -a"` and then `tail -f -n0 ~/.bash_history`, but this doesn't capture ssh, subshells, and only shows the command after it is completed.

Better yet still simple: Many Software Carpentry instructors use this script, which sets the prompt, splits the terminal window using tmux and displays command history in the upper panel. Requirement: tmux

Better (bash): This prints the output before the command is run, instead of after. Tail with `tail -f ~/demos.out`.

```
BASH_LOG=~/demos.out
bash_log_commands () {
    # https://superuser.com/questions/175799
    [ -n "$COMP_LINE" ] && return  # do nothing if completing
    [[ "$PROMPT_COMMAND" =~ "$BASH_COMMAND" ]] && return # don't cause a preexec for
↪$PROMPT_COMMAND
    local this_command=`HISTTIMEFORMAT= history 1 | sed -e "s/^[ ]*[0-9]*[ ]*//"`;
    echo "$this_command" >> "$BASH_LOG"
}
trap 'bash_log_commands' DEBUG
```

Better (zsh): This works like above, with zsh. Tail with `tail -f ~/demos.out`.

```
preexec() { echo $1 >> ~/demos.out }
```

Better (fish): This works like above, but for fish. Tail with `tail -f ~/demos.out`.

```
function cmd_log --on-event fish_preexec ; echo "$argv" >> ~/demos.out  ; end
```

Obselete: The below commands rely on recording your entire session using `script`, and then dynamically following the output. This allows you to track commands even in subshells/over ssh, but introduce a lot of other errors in corner cases. These might work but needs debugging (there are lots of complexities in extracting out the right parts). Note: some of these ignore the first line you type.

```
script -f demos.out

# most general... prompt must end in '$ '.
tail -n 0 -f demos.out | awk '{ if (match($0,/^[^$ ]+ ?[^$ ]*[$][[:cntrl:]0-9m;[]{,10}
↪ (.*)/,m)) print m[1] }'

# Prompt format of [username@host]$
tail -n 1 -f demos.out | while read line; do [[ "$line" =~ \]\$\ ([^ ].+)$ ]] && echo␣
↪ ${BASH_REMATCH[1]}; done

# Standard bash prompt of 'user@host$ ' (less likely to have false positives)
tail -n 0 -f demos.out | awk '{ if (match($0,/^[^@]+@[^$]+[$][^ ]* (.*)/,m)) print␣
↪m[1] }'
```

(continues on next page)

(continued from previous page)

```
# Prompt is $ ' alone on a line.
tail -n 0 -f demos.out | awk '{ if (match($0,/^[$] (.*)/,m)) print m[1] }'
```

```
# used for the fish shell (note: untested)
tail -f -n 0 ~/fish_history | sed -u -e s'/- cmd:/ \>/'

# used for zsh shell (put this into a script file)
clear >$(tty)
tail -n 0 -f ~/.zsh_history | awk -F\; 'NF!=1{printf("\n%s",$NF)}NF==1{printf("n %s ",
↪$1)}'
```

5.3 Starting off

- Don't start off with tech details, say why this is important. Think of what the emotional ("coolness") appeal is and start off with that.
- Why will this be useful?

5.4 Team teaching

- Discuss with co-teachers and helpers about what each of you will do.
  - Hand signals for common situations: too fast/slow in general, louder, time for a break, "good enough, move on", "explain more here".
- It can be hard for one person to manage everything. How can multiple instructors take part? Probably the most common ones are:
  - Teach teaching: alternating
    - Commander and navigator: conceptually divide roles of big picture teaching and doing the details.
    - If "real" alternating, each section should be 10-15 min at least, otherwise too much context switching is distracting.
  - Teach and assist (master helper going around)
  - Teach and observe.
  - Asking directed questions to fill in gaps.
- Tell the students the way the teachers will work together, so that it seems coordinated rather than someone is interrupting.

5.5 During the lessons

- Helpers can read *the helping and teaching guide*. Encourage helpers to stand and be constantly walking around, people rarely flag helpers from across the room.
- Encourage the use of sticky notes (red=need help, green=I am done with the solution). They can also be used for voting, e.g. red/green for two answers of a multiple choice question.
- Don't touch the learner's keyboard! This is very hard to do, since it's only natural to want to get things done quickly. The best idea we have is to have a pen and sticky notes, when it's hard to spell out a command to type, write it instead.
- If appropriate for your topic, create a cumulative cheatsheet/diagram on the board as you are presenting.
- Take advantage of the mistakes/typos you make when teaching! When you do a mistake and get an error message and realize what you did wrong, explain what happened since this can offer valuable insights to learners.
- Ask "do you do X?" where X is what you are teaching. Instead, ask "how do you do Y?". The first question implies something you are doing wrong, the second is open-ended.

5.5.1 Try to stick to the material

- Don't try to show everything, show less, but show it clearly.
- Try not to completely deviate from the material. Ideally, rather influence the material before you teach. Of course it is good to react to questions and to adapt the material to the audience, so sometimes an excursion can be very useful, but make clear that you then deviate from the script and be explicit about whether participants should follow what you do or only watch.
- If you want to show some extra steps in the terminal, show them perhaps at the end of an exercise block to not "mess up" the exercise half-way and change it with respect to the material.
- It is good to mention an anecdote or two but be careful about mentioning too much new jargon which only very few participants may relate to.

5.6 Wrap up

- Say what you taught and why.
- Say what comes next. Say where to get that.
- Update the instructor's guide and file issues for any problems you noticed.
- Use the sticky notes to get good/bad feedback: have people write one good and one to be improved thing, and leave the note on the door on the way out.
- Get instant feedback from your co-teachers and helpers (students too, if they offer any).
  - Consider making notes on a 4-way diagram of (content$\leftarrow\rightarrow$presentation) $\times$ (went well$\leftarrow\rightarrow$can be improved).

CHAPTER

SIX

SUMMARY OF TEACHING TECH TOGETHER

Teaching Tech Together is a book compiled by Greg Wilson which is about the pedagogy and practical hints of teaching technology in informal environments. It is a very good resource, and the main point is that research does back up teaching, it's not all intuition. Many citations are included.

This page contains a summary of the most important points. The point is that one can quickly refer to this before writing a new lesson or teaching a course. The article Ten quick tips for creating an effective lesson is also a good summary of the main lesson design points of this book.

Useful appendices:

- Backwards lesson design template
- Checklist for events

6.1 Ch1: Introduction

- **Novice = no good mental model of what they are learning**, "not even wrong"
- A manual is *not equal* to a tutorial - a tutorial needs to build a mental model from scratch.
- Formative assessment = determine what the misconceptions are.

6.2 Ch2: Building mental models

- "Expert blind spot" = experts have more links, so don't see what links are missing.
- Concept maps as a metaphor for connections
- 7+/-2 concepts can fit in short term memory at once
- Get feedback from others, then give feedback to others, then self-feedback (last one is "deliberate practice")

6.3 Ch3: Expertise and memory

- Cognitive load: too much is bad and makes learning slow
- **Faded example**: blank out certain things in an example which are added as an exercise/example (what you want to progressively teach). Seeing examples is good, debugging as an example.
- "I want to do something, not learn how to do everything"
- **Parsons problems** - give working code but in random order, students must put it into the right order.
- Minimal manual: one page micromanuals on specific tasks. Helps training but loses content.
- The last exercise of this chapter has some good hints for making useful graphics.

6.4 Ch4: Cognitive load

- Cognitive load is divided into intrinsic load (background required learn), germane load (mental effort to link new to old), and extraneous load (everything else that distracts from learning). (this is "cognitive load theory")
- A paper claimed that self-guided learning is less effective, because people are overloaded: you have to both learn new facts and learn how to use them at the same time.
- Strategies: use exercises well that minimize tho load. a) parsons problems, b) labeled subgoals, c) split attention (separate channels, but complimentary rather than redundant), d) minimal manuals

6.5 Ch5: Individual learning

(chapter about how people can help themselves)

- Six strategies: a) spaced practice, b) retrieval practice, c) interleaving (abcbac better than aabbcc), d) elaboration (explain to self), e) concrete examples, f) dual coding (e.g. words and pictures, or different forms of same material).
- Manage time well
- Peer assessment

6.6 Ch6: A lesson design process

- Backwards lesson design, similar to test-driven development: 1) brainstorm ideas for what to cover, 2) create learner personas to figure out who you want to teach, 3) create formative assessments to give learners a chance to exercise what they are trying to learn (3-4 per hour), 4) put formative exercises in order, 5) write the teaching material around this.
- Learner persons, to guide your design process: a) general background, b) what they already know, c) what they *think* they want to know, d) how course will help, e) special needs.
- Learning objectives: write objectives and think of what depth of understanding you are getting too. Consider Bloom's taxonomy: a) remember, b) understand, c) apply, d) analyze, e) evaluate, f) create.
- Fink's taxonomy (unlike Bloom's, complimentary not hierarchical): a) foundational knowledge, b) application, c) integration, d) human dimension, e) caring, f) learning how to learn.

- Maintainability: is it easier to update than replace? a) **You have to document the lesson design process**, b) technical collaboration, c) are people willing to collaborate? Or do teachers resample rather than update?

6.7 Ch7: Actionable approximations of the truth

(chapter about learning programming specifically… title comes from not necessarily having clear research that says what you *should* do, but you have to do something anyway)

- Experts know *what* and *how*, novices lack both but most teachers focus on *what* only.
- Think about teaching debugging and using it as examples - the *how*.
- If you are teaching programming specifically, just read the chapter.

6.8 Ch8: Teaching as performance art

- Get feedback on your teaching. People aren't born teachers, and feedback isn't in the western teaching culture enough.
- Use live coding. It's much more effective, especially because it's two way and *you can demonstrate making mistakes*. a) embrace your mistakes, b) ask for predictions, c) take it slow, d) be seen and heard (stand + microphone), e) mirror your learner's environment, f) use the screen wisely (make it big enough), g) double devices (one to present, one for notes), h) use diagrams, i) avoid distractions, j) improvise after you know the material, k) face the screen only occasionally
- Drawbacks of live coding, which you can minimize over time: a) going too slow, b) exercises can be too deep and have too much cognitive load (give skeleton code).

6.9 Ch9: In the classroom

- Code of conduct: teaching isn't for those that are already "in", it's for those that aren't. If you don't notice problems and enforce it transparently, it means nothing though.
- Peer instruction. Discuss in groups. e.g. multiple choice question, if there is a wide variety of wrong answers, have them discuss in groups.
- Teach teaching: different strategies, consider what you want to do: a) teach teaching (taking turns) b) teach and assist (going around helping) c) alternative teaching (group with more specialized instruction), d) teacher and observer, e) parallel teaching (two groups, same material), f) station teaching (rotate through stations).
- If co-teaching, plan ahead: a) confirm roles at start, b) work out some hand signals for common conditions, c) each person should talk at least 10-15 min at a time, d) person who isn't teaching shouldn't distract, though leading questions OK, e) check what your partner will teach after you are done, f) inactive teacher stays engaged, not doing own stuff.
- Plan for mixed abilities, especially false beginners who have studied the material before.
- Can you make a collaborative not online document?
- Sticky notes
- Don't start from blank pages, give some starting point. Many other good points in the chapter itself.

6.10 Ch10: Motivation and demotivation

- Extrinsic vs intrinsic motivation. Extrinsic: have to do it for job or something. Intrinsic: do it for self, you want to encourage intrinsic motivation. Drivers of intrinsic motivation: a) competence, b) autonomy, c) relatedness (connection to others).
- Consider usefulness and time to master. Focus on useful and fast. Useful = *authentic tasks*, things people will actually use.
- Avoid demotivation: for adults, a) unpredictability, b) indifference, c) unfairness. Specific examples: a) contemptuous attitude, b) saying existing skills are worthless, c) complex or detailed technical discussion, d) pretending you know more than they do, e) the word "just" as in, it's "just easy", f) software installation problems, g) giving impossible challenges to fail at to try to learn something, if not understanding.
- Consider accessibility and inclusivity - consider things are harder for others, try to understand diversity of backgrounds.

6.11 Ch11: Teaching online

- Disadvantage of MOOCs: can't clear up individual misconceptions
- The chapter has various good ideas, including how to make sure everyone is heard (certain group doesn't dominate online discussions), short cycles and short exercises, require some small group work, use videos to engage rather than instruct (people can read faster), identify and clear up misconceptions early.
- Flipped classroom: watch lectures on own time, do exercises and discuss in class time.

6.12 Ch12: Exercise types

- Multiple choice, code yourself, code+multiple choice, inverted coding (given code, test and debug), fill in the blanks, Parsons problems (given questions but in wrong order).
- Tracing execution, tracing values, reverse execution (find input for output), minimal fix, theme and variations, refactoring exercise. Pen and paper exercises.
- Diagrams and connection: draw diagram, label diagram, matching problems.
- Autograding is hard, in particular most automatic grading tools don't provide useful feedback messages. Also, automatic grading can only test low-level skills, not higher abstractions like code review.

6.13 Ch13: Building community

(Chapter about forming a community of teachers and learners working together)

6.14 Ch14: Marketing

- Think about what you are offering to who. Who are the target audiences and why should they be care and become invested?

6.15 Ch15: Partnerships

- Main two points are work within schools or outside of schools. If inside, part of academic programs? Academic programs and especially teachers change very slowly.

6.16 Ch16: Why I teach

(A note from the author)

CHAPTER

SEVEN

WORKSHOP REQUIREMENTS - IN PERSON

This checklist is for the pre-planning phase of in-person CodeRefinery workshops: where you are deciding if you can host one and what room to use. Let us know about the items on this list when you contact us.

7.1 Lecture room

- The room needs to be sufficiently large (a typical workshop is attended by around 20 learners and 4 instructors).
- There needs to be enough space for instructors to walk around and interact with learners individually (a "flat" room is required).
- Learners should face the same direction, and learners should be able to sit side-by-side for pairwise work.
- The room should preferably have windows, and be ventilated well enough so that 20-30 people (and same amount of laptops) will not make it too warm.
- A coffee room (or similar) should be located nearby for the coffee breaks.
- Two overhead projectors are desirable, but if only one is available that will work too.
- The projector screen needs to be large, and the resolution of the projector needs to be good.
- Stable wireless connectivity for 20-30 people.
- Sufficiently many electricity outlets so that all participants can charge their laptops.
- Standing board for instructor.

7.2 Helpers

CodeRefinery workshops are hands-on and interactive, and a lot of time is spent on exercises where participants learn by doing. Participants explore themselves, and that means they need guides to help them if they get stuck.

We recommend that each site takes proactive steps to recruit at least two helpers per workshop. We've noticed that helper diversity promotes learning, so we recommend that organizers also make proactive steps to have diverse helpers (male/female, international, etc.). Local organizers should directly contact possible helpers and invite them.

Good candidates are people who have any of:

- have attended a previous CodeRefinery workshop
- have a passion for teaching, scientific software development, open source, open science, etc.
- are research software engineers or hold a similar technical research position
- have experience from teaching e.g. Software Carpentry workshops

- want to experience CodeRefinery but already have a good idea of most basics

7.3 Other requirements

When we organize a workshop or event at a new site, we may need help with some local arrangements, including:

- Booking a lecture room.
- Ordering coffee and refreshments.
- Advertise the workshop through local dissemination channels.

7.4 After the workshop

Would you like to become a helper, instructor, or partner and make more workshops possible?

CHAPTER

EIGHT

ORGANIZING A CODEREFINERY WORKSHOP

Anyone can organize a CodeRefinery workshop and teach the CodeRefinery lessons which are licensed under CC-BY. However, making it a successful workshop requires careful planning and preparation. Here we will go through practical aspects of organizing a workshop.

8.1 Select a workshop coordinator

One or two persons coordinate the workshop preparation and debrief. This does not mean that they do all the work - they are encouraged to delegate tasks - but they make sure that nothing gets forgotten.

8.2 Other documents and references

- Workshop organization overview: https://github.com/orgs/coderefinery/projects/4
- Instructions on how to set up a registration page in Indico (for NeIC affiliated staff): *indico-workshop-management.md*
- Email templates for workshop communication:
  - 1-2 weeks before workshop starts
  - advertising workshop via private communication

8.3 Before the workshop

8.3.1 First steps

- Recruit instructors - having at least 3 instructors is highly recommended.
- Find 1-2 workshop helpers *with an appropriate background*.
- Reserve dates (coordinate this with the instructors)
- Reserve room
- Select a workshop coordinator
- Workshop coordinator creates a ticket with a checklist on https://github.com/orgs/coderefinery/projects/4 and takes it (self-assigns)

8.3.2 Lecture room

- Start looking for an appropriate lecture room early.
- See this *list of requirements* for the lecture room.

8.3.3 Set up workshop page

- Import the template at https://github.com/coderefinery/template-workshop-webpage to your username or the coderefinery organization, and name it like "2019-10-16-somecity".
- Update the required fields in `index.md` and push the commits. The page should now be served at *username.github.io/2019-10-16-somecity/*.
- If the workshop will be customized to the needs of a particular audience, modify the schedule accordingly.
- If the workshop should be listed on https://coderefinery.org:
  - (Fork and) clone https://github.com/coderefinery/coderefinery.org
  - Under `coderefinery.org/_workshops/`, add a file named like `2019-10-16-somecity.md` which contains the fields permalink, city and dates. For example:

```
---
permalink: https://username.github.io/2019-10-16-city/
city: Somecity
dates: October 16-18, 2019
---
```

  - send a pull request with your new file.
- Create a registration form following *indico-workshop-management.md*.
- Open and test registration

8.3.4 Announcing the workshop

- Twitter
- Email persons who registered to notify-me form
- Use local mailing lists and all channels possible

For self-organized workshops:

- Write an email to support@coderefinery.org to get a pre-workshop survey link and registration form on https://indico.neic.no

8.3.5 Distribute the work

- Make sure lessons are distributed

8.3.6 Preparing lessons

- Go through the lesson material you will be teaching and think about how you intend to teach it, and how much time you will be spending on each episode.
- Are there any unsolved issues that you can fix?
- Go through the instructor guides of the lessons you will be teaching.
  - Review the intended learning outcomes, and try to keep these in mind while teaching.
  - Try to memorize the typical pitfalls and common questions.
- Go through the lesson presentation hints.
- Go through the helping and teaching guide, and request all helpers to go through it too.

8.3.7 Prepare practicals

- Order catering (coffee, tea, water, fruit, something sweet, etc.)
- Organize sticky notes
- Organize extension cables if needed
- Organize alternative wireless for those without Eduroam (if any)

8.3.8 Communication with participants

- Send out practical information, including installation instructions, around 2 weeks ahead. Here is a template.
- Emphasize that all software should be installed before the workshop starts, and point out the configuration problems and solutions.
- Remind registered participants that they are either expected to show up or to cancel participation
- Also ask those without Eduroam to speak up.
- Maintain waiting list if needed
- Make sure we have enough pre-survey answers
- Close registration on the workshop page

8.3.9 1-2 weeks before the workshop

- Workshop coordinator organizes a call with all instructors and helpers to discuss the schedule to leave no doubts about timing. Also discuss the survey results.
- Point helpers (and instructors) to the *tips for helpers*.

8.3.10 Right before the workshop starts

- Prepare a shared Google doc or https://hackmd.io with global write permissions, consider creating a memorable short-link (e.g. bit.ly)

8.4 Create exercise repositories

- The collaborative Git lesson requires exercise repositories to be set up. For this follow the instructor guide in the lesson material.

8.5 As participants arrive

- Emphasize to participants that you need to sit with someone - don't work alone.
- Try to have participants sit next to someone with a similar operating system if they have no preference, since they will face similar problems.

8.6 Introduction talk

- See https://github.com/coderefinery/workshop-intro
- Have a 10 minute ice-breaker session where participants and instructors introduce themselves and either describe their research in 2-3 sentences or what they hope to get out of the workshop.

8.7 During workshop

- While teaching, keep these tips in mind
- Don't start off with tech details, say why this is important.
- Try to stick to the material, although some excursions are useful.
- Keep up interactive feel by encouraging and asking questions
- Keep time
- For presentations which have shell commands, create a cheatsheet/reference on the board in real time.
- Remind participants about sticky notes.
- Make sure we take regular breaks (at least a short break each hour)
- Give participants some time to also experiment (do not rush the classroom through exercises)
- Encourage optional feedback at the end of each day or end of each lesson on sticky notes. Process the feedback immediately and adjust your teaching (pace etc) accordingly
- Create GitHub issues for points which are confusing or problematic
- Take active part even in the lessons you're not teaching, e.g. by asking questions and (politely) interject with clarifications when you think something is confusing to the learners
- Wrap up, say what you taught and why, and what comes next.

8.8 At the end of workshop

- Give credit to those who contributed and helped
- Use https://github.com/coderefinery/workshop-outro

8.9 Post-workshop

- Process and distribute feedback to co-instructors and others (e.g. type up in shared document)
- Debrief with instructors
- Process certificate requests

8.10 Post-workshop survey

To measure the long-term impact of CodeRefinery workshops it's useful to send out a post-workshop survey. This survey can identify which topics taught in workshops are particularly useful and which have less benefits for the participants.

CHAPTER

NINE

ONLINE TRAINING MANUAL

Also please read our lessons learned.

This manual covers general guidelines for conducting online training as well as specific tips on using Zoom.

9.1 For the instructors

If you have an old spare laptop, connect to the call as a second "you" and you can watch and verify your screensharing and fontsize to avoid "Am I sharing the screen? Hopefully you see what I see."

9.2 How to avoid "Zoom bombing"

- Either set a password or use waiting rooms
- Share connection details only with participants and helpers, not on the web
- Disable file transfer
- Disable "Allow removed participants to rejoin"

9.3 Preparation

- Schedule the meeting/webinar in the online Zoom system
- do not auto-mute participants' microphones, as this also happens when you enter breakout rooms.
- Decide roles:
  - Decide the Zoom host and co-hosts
  - Use panelists? (Zoom webinar feature)
  - Decide instructor and backup-instructor in case of network issues
  - Decide helpers. One helper should be responsible for monitoring Zoom, i.e. the chat window, hand-raising and other feedback
- Co-hosts, breakout rooms and feedback controls need to be enabled (on website) before the meeting starts. If options are reconfigured, the meeting may need to be ended and restarted for them to take effect.
- Create enough breakout rooms at the beginning since this cannot be easily changed during the meeting.
- TODO: set up pre-lesson polling? (zoom feature) Maybe unnecessary in view of pre-workshop survey

- Instructors and helpers should use a reliable camera and microphone. Computer microphone might not be enough since audio quality will depend on instructor's head angle and proximity to screen.
- Workshop owner creates a HackMD which will be used for collaborative note taking.

9.4 At the beginning of the session

- Allow time at the beginning of the session to debug video/audio and to arrange windows. This takes few minutes so better do not start with teaching from minute 1. Plan for an early 5-minute break to debug this.
- We cannot assume that all Zoom participants have the same and up to date client and some clients do not contain "sticky notes" feedback or a button to raise hands so agree with participants on signals (e.g. typing `\hand` in the chat window seems to be standard).-
- We demonstrate how HackMD works and use it in an ice-breaker (roll call or asking a questions).

9.5 Recording of sessions

If you plan on recording and publishing the session, prepare in advance so that you don't have a difficult editing job later. Make sure that you (or users) don't show any personal or confidential information. Think about what happens if users speak: do you ask for permission to publish in advance (maybe encouraging people not to), or edit it out later (taking your time later).

If you plan to record the session, make sure that everybody is aware that the sessions is recorded, informed about how the recording will be used, and gives consent to be recorded: https://support.zoom.us/hc/en-us/articles/360026909191-Consent-to-be-Recorded

In Zoom it's important to start recording in the form you want the video to be in (e.g. start recording when screen is shared so that it stays there): https://support.zoom.us/hc/en-us/articles/360025561091-Recording-layouts

Set screen background to black. We saw a glitch in Zoom which caused the background image to flash above the screen, if it was pure black it would be less distracting.

9.6 Zoom-specific installations instructions sent out before workshop/lesson

- Recommend to install Zoom app. Browser is possible but more limited
- Test-launch zoom and test microphone, speaker and camera (lower left corner buttons)
- Instruct participants to watch a zoom introduction (TODO: insert link), and play around with zoom.us/test to get acquainted with interface.
- Optional: set up virtual background
- "During the workshop, you might be asked by a helper to share your screen. Make sure to keep private information away from the screen you share."

9.6.1 Contingency plans

- Be prepared for intermittent network problems.
- There should be a backup instructor in case the main instructors disconnects
- Learners might occasionally experience lag and temporary network hickups. This makes it particularly important to speak slowly and repeat important topics.

9.7 Breakout rooms

- Breakout rooms can be used both by helpers to assist individual learners during an exercise, or for multiple learners working on a group exercise.
- When creating groups, the host or co-hosts can choose automatic setup, where only the number of groups is selected and the distribution into groups is automatic, or manual setup where the host/co-hosts distribute learners into groups.
- Host needs to move helpers, co-hosts cannot enter rooms on their own.
- Somebody asking for help gets assigned to a room together with a helper.
- TODO: is it possible to create breakout room for only some participants, leaving other learners unaffected? This is crucial for helping participants during exercises who have raised their hand. Need to test this
- Host and co-hosts can join any room and jump between rooms. This should be used during collaborative exercises to see how the exercise is progressing or participate in the group work.
- When a collaborative exercise is about to end, the host/co-hosts can broadcast a message into all groups.
- When the host/co-hosts end a breakout room session, participants in groups have 60 seconds to finish before the session terminates.

9.8 Exercises

- Just like in a regular workshop, demonstrations and type-along sessions should be interspersed with frequent exercises
- For pairwise or group work exercises, the instructor (or Zoom assistant) should create breakout rooms with chosen number of participants in each
- For single-person exercises, no breakout rooms are needed
- Learners should be instructed to raise their hand when they need help. This corresponds to putting up a red sticky note in in-person workshops.
- TODO: what signal should be used for green sticky notes?
- Polling can be used as formative assessment questions. The host creates a poll based on a lesson template and requests learners to answer. (TODO: polling seems not available in kth-se zoom subscription)

9.9 Breaks

Following an online event can be even more tiring than a physical event and therefore also during online sessions we need to plan for breaks as we would for an in-person event.

9.10 More resources

- https://foundation.mozilla.org/en/blog/tips-make-your-zoom-gatherings-more-private/

CHAPTER

TEN

GUIDE ON HOW TO USE INDICO FOR MANAGING WORKSHOPS

We use the NeIC Indico service, https://indico.neic.no/, so you need to create an account at https://indico.neic.no/login/.

Radovan, Thor, and Sabry are managers of the CodeRefinery category in indico.neic.no and will need to grant you permissions to create event pages.

To create a new workshop page, it is easiest to clone a previous event. This copies the registration form and metadata, but not the pre-workshop survey which needs to be manually imported as a json file.

10.1 Step-by-step instructions:

10.1.1 Copy basics from latest event

- Visit https://indico.neic.no/, and click CodeRefinery which takes you to https://indico.neic.no/category/5/.
- Click the latest workshop event. You might need to show "events in the future" to see the latest event.
- Go to admin mode (click the pen symbol on top toolbar, "Switch to the management area of this event").
- Click the "Clone" button, and select "Clone Once". Click "Next" button.
- For "What should be cloned", select "ACLs and protection settings" and "Registration forms". Click "Next".
- Confirm category "CodeRefinery", and click "Next".
- Select the start date and time of the workshop, click "Clone".
- You are now on the cloned event page (confirm that the event number changed), and you should start updating the information.

10.1.2 Update copied event information

- Update the Title, Description, Date, Time, Room, Venue and Address fields by clicking the pen symbols on the right.
- Click "Registration" from the left-hand menu, and confirm that there's a registration form, probably with a wrong title.
- Click the "Manage" button on the "List of registration forms",
  - Click "Edit" on the "General settings"
  - Update the registration form name and both the fields "Contact info" and "List of recipients" with your own email address to get notifications on new registrations.

  - **Waiting list:**
  - *Indico doesn't have an actual waiting list functionality. To implement a waiting list, we use moderated registrations and confirm all registrations up to max capacity (eg. 40). Registrations after that up to maximum number of participants (eg. 60) are left unconfirmed and an email is sent manually from Indico to the registrant that he/she is on the waiting list. Now we have a waiting list of size 60 - 40 = 20.*
  - Activate "Moderated" which will require each registration to be approved.
  - Set maximum number of participants (after which registration is closed), this should be *room capacity* + *waiting list size*. Click "Save".

More information about the registration process

- The Description field in the general settings should contain additional information about the registration process:

```
Welcome to the registration page for the CodeRefinery instructor training␣
↪workshop in Stockholm!

To complete your registration, you need to:

1) Enter your registration details by clicking the "Apply now" button below.
2) Fill in the pre-workshop survey by clicking the "Fill out the survey" button␣
↪below.

Confirmation email
After filling out the registration form you will receive an automatic␣
↪confirmation email but please note that your registration is only tentative␣
↪until we confirm it with another (human-written) email which should happen␣
↪typically within a week.

Waiting list
We maintain a waiting list for seats but this is currently not automatic so we␣
↪need this short time buffer to manually confirm participants and inform those␣
↪who are on the waiting list.

First come, first serve
The seats are assigned on a first come first serve basis but we need to also make␣
↪sure that registered participants are affiliated with a Nordic academic␣
↪institution since the course is free for participants and financed by the␣
↪Nordic e-Infrastructure Collaboration (unless this is an event outside of␣
↪Nordics funded by a different organization).

Cancellation
We ask confirmed participants who are not able to participate at the course they␣
↪have signed up for, to inform us as soon as possible so that people on the␣
↪waiting list can take the vacant seat.

Questions?
If you have any questions about your registration status, please write to␣
↪support@coderefinery.org.

Looking forward to seeing you at the workshop!
```

10.1.3 Import survey

- Now click "Surveys" from the left hand menu. You will now import the standard pre-workshop survey from a json file.
- Go to https://github.com/coderefinery/pre-workshop-survey and clone the repository.
- Go back to the Indico Surveys page, and click "Create survey"
- Name the survey "Pre-workshop survey", enable the option "Anonymous submissions" and disable "Only logged-in users". Click "Save".
- Back on the "Surveys" page, click "Manage" on the newly created "Pre-workshop-survey" survey.
- It will say "Survey not ready". Click "Prepare questionnaire".
- Click the "Import" button, click "Choose from your computer", and find the file exported-survey.json" from the pre-workshop-survey repository you cloned. Click "Save".
- Go back to the survey page (click "Surveys" on the left), and click "Manage". Click the "Open now" button to let the survey go live.

10.1.4 Open registration

- Go to the Registration page from the left-hand menu, and click "Manage".
- It will say "Registrations are not open yet". Click "Start now" to open for registrations.
- Click the blue "Switch to display view" on the top left.
- Confirm that both the "Surveys" and "Registration" links can be seen.
- Click both links to do a test registration
- Once you manage to test-register, update the workshop webpage, and announce via Twitter.

10.1.5 Exporting registrations

- Go to the Registration page from the left-hand menu, and click "Registrations" which takes you to the list of registrations..
- Click the check-box on the menu just above the list of registrations and select "All".
- Click on "Export" from the top menu, select "CSV" and choose a download directory.
- You can use the read_csv.py to parse the CSV file and print selected fields, e.g. email addresses to be used in sending out information to participants.

CHAPTER

ELEVEN

ICEBREAKERS

This is a list of possible icebreaker questions.

You should make it very clear that *everyone* should answer the question, and thus it should be very broad. The point is to make sure they know how to use the tools. Make sure that the question feels inclusive - not just that people can answer, but that it doesn't make people feel they are far behind others.

If you ask people to add there name as part of an introduction, the document becomes personal data and must be controlled more, and sets you on a path to extensive editing before it can be released. Think before you do this - maybe you just ask for information about backgrounds without names.

11.1 Relevant to workshop

An icebreaker isn't supposed to be relevant to the workshop, but it could be useful some days or as a second question.

- What from this workshop are you going to use in the near future?
- What was the most confusing thing from yesterday?
- Have you already used what you have learned in the course during your work? If so, what?
- What is the most useful thing you know, that you wish someone had just told you about computing when you first started it?

11.2 General

- What's a good icebreaker question?
- How is the weather where you live?
- How are you doing?
- Are you happy to continue this workshop for another week?
- Is that an Iphone?
- If you could have anything what you want for dinner today, what would it be?
- What cool thing/tool have you discovered/learned the past days? (independently of this course)
- Is this course part of your work? Or do you spend free time on it?
- Do you like olives?
- Are you annoyed at the size of anaconda?
- What's your favourite pizza?

- Do pineapples :pineaplle: belong on pizza?
- What did you have for breakfast?
- Do you like Python?
- Where's your favorite place to nap?
- Do you use git or identify as one?
- When and how did you learn to program?

11.3 Credits

Most of these questions came from a "What is an icebreaker" question in the first Mega-CodeRefinery workshop.

CHAPTER

TWELVE

LESSON DESIGN

This is a checklist and hints when writing and designing a new lesson. The master material is in Teaching Tech Together, primarily chapters 6 and 12 for practicalities and 2 and 4 for big picture considerations. But really, all the book. See *the summary we made* or the actual book. The article Ten quick tips for creating an effective lesson is also a good summary of the main points. Finally, the Carpentries Curriculum Development Handbook gives practical information on how to design a new lesson and covers the entire lesson life-cycle with a good overview of the lesson release timeline.

This doesn't replace your own knowledge in doing the actual teaching part. Instead, the first half gives pointers on making sure your audience can connect to the material, and the last half gives hints to help you come up with good exercises and examples.

12.1 Backwards lesson design

Think test-driven development: decide what you want students to be able to do, design exercises to measure it, then fill in the gaps with teaching. You can see their summary. The steps are:

1. Brainstorm what you want to cover.
2. Create or reuse learner personas - understand who you want to teach. What do they care about? Perhaps as important is what they don't care about: make sure that you don't go too in depth too early and turn people off.
3. Create some summative assessments, that show what learners should learn by the end. Try to connect these to the learner personas.
4. Create formative assessments (exercises) that let the learners practice what you want them to learn. See below for hints on coming up with good exercises. These should also connect to things the learners will actually do, but can also be more of checkpoints.
5. Put exercises in a logical order, and fill in any gaps. Ideally there should be 15-20 min of teaching between each exercise. Perhaps most are short (a few longer examples as needed), to identify a certain learning goal and misconception.
6. Write just enough material to get from one exercise to the other.

The most important point here is to start from learner's needs and how they can feel connected, not from the tech details.

When advertising the course, connect it to your learner personas so that you get the right audience and they know why they should come.

12.2 Emotional and intrinsic appeal, other basics

You can think of why people should feel emotionally connected to your material - maybe it's too much to expect people to get emotionally invested, but if you try for that, you'll end somewhere better.

Try to design around tasks and exercises which your audience will care about. For example, don't say "here are some shell commands", but "aren't you tired of copying all of these files one by one... check out the shell... once you know it, you will really feel at home. Here are some typical things you might do.". Intrinsic motivators include **sense of agency** (being able to do things themselves), **competence** (usefulness of what they are doing, feeling they know something), and **relatedness** (doing things that others are doing).

A manual is reference, a tutorial builds a cognitive model. If you can build the cognitive model and tell them the "why", students may be able to refer to the manuals themselves and become self-sufficient. Thus, teaching should be more of a tutorial, with good links to manuals (it can also explicitly teach how to use the manuals).

Perhaps a related point is inclusiveness: make sure there's not some "in" crowd. Perhaps the best description I have seen: don't assume that some people are missing something, but that others have had the fortune of learning it earlier. This may not matter in a purely factual lesson design, but if you are trying to make things intrinsically or emotionally appealing, it is essential.

12.3 Who is the audience?

Making the **learner personas** are essential to making a good lesson, even if you think you know who you are teaching to. This is because it grounds you into what your audience already knows (or doesn't know) and what they are interested in.

You also have different ways people can refer to the material:

- In a class, with an instructor guiding them
- Reading along by themselves
- In a class, being much more advanced than others, so that they skip ahead and do advanced material themselves.

12.4 Planning

- Do some planning, and *document it* - the design process helps others to teach and modify. At least put it in the README. (this is the **designer/maintainer's guide**)
  - Put the main points from the "backwards lesson design process" in here, enough that it is easier for someone to improve your lesson than to redo it.
- Make learner personas: what is your target audience?
- Decide learning objectives based on the personas: high-level end goals. What students get out, not what they do.
- Also make a *guide* for teaching (**instructor's guide**), "if you want to present this, do this".
  - How much preparation is needed? Is it enough to know the topic and have read the material?
  - Things to prepare before the presentation. Does anything need to be set up?
  - Practical notes on presenting.
  - Are there solutions to exercises somewhere? Are they needed?
  - Include some pre-assessment questions which can be asked at the beginning.

  - Perhaps you should do this at the end, but at least starting the instructor's guide at the beginning will frame your writing.

12.5 Writing

There is not much here yet, mostly just follow the "backwards lesson design" above. The hardest part is coming up with good exercises, so our practical advice is to mix and match from the two taxonomies at the bottom and the exercise types. Try to think of diverse types of exercises.

Exercise design is the time it is most useful to be with others to do brainstorming, so we highly recommend discussing with others at this point. Because exercises are used to set the overall outline of the lesson, this also gives people a say in the overall outline - in a very concrete way.

- Make sure you include the emotional starting point at the beginning - why should you care and why is this cool?
- This should also be at the start and end of each section: not just what or how, but why?
- Part of this is also having a **student's guide**, so that people independently studying can know how to follow the material.
- It's OK to have more material than can be presented or than people should know, but *label things well*, including *labeling the difficulty*.
  - In the beginning, what sections are expected to be taught in short/long versions? What's advanced/optional?
  - Label advanced and optional sections as such. Perhaps also really basic sections that can be skipped for that reason.

Plan for mixed abilities. It's OK to have optional (basic) and advanced sections, as long as they are clearly labeled. Mainly, don't have people think that you are uncoordinated because you are skipping advanced sections.

Once you are done, update maintainer's and instructor's guides.

12.6 Introduction (and conclusion)

The introduction is the first thing people hear, and needs special thought. Don't start with a cold open, just going straight to the topic ("what" or "how"). Instead, have some careful motivation ("why"). It could be especially good to talk about what is wrong with the current state of affairs (give a good, simple example) and why it should be improved. Then start talking about what the improvements are.

Ideally, the introduction should serve as an self-contained abstract of your material. If you need to teach your lesson in only 10% of the time you have, can you use just the introduction to do it?

Conclusion should remind people about why this is cool and discuss what comes next.

12.7 Thinking of exercises

Not every exercise has to be an amazing hand-on example. It's mixing with smaller, more conceptual things to reduce the cognitive load and be able to have more frequent exercises.

One of your other primary goals should be to make your exercises *relevant*. Abstract will lead to disconnection. Connect the exercises to the real world. Also, can you tell a complete story with exercises? (Remember, in backwards lesson design, the exercises form the story of the lesson.)

Remember that not every exercise has to be long. Try to have frequent short exercises to get immediate feedback, with some long ones.

Good exercises are the most important factor in a good lesson. Even if you are preparing the rest of the lesson mostly alone, consider a good long brainstorming session to go from "list of topics to cover" to "sequence of exercises".

When you are stuck thinking "how can I make an exercise that covers X", think of the lists below inspiration. Not every exercise has to be an sophisticated hands-on thing, so don't be afraid to use different types:

Basic types:

- Multiple choice (easy to get feedback via a classroom tool - try to design each wrong answer so that it identifies a specific misconception).
- Code yourself (traditional programming)
- Code yourself + multiple choice to see what the answer is (allows you to get feedback)
- Inverted coding (given code, have to debug)
- Parsons problems (working solution but lines in random order, learner must only put in proper order)
- Fill in the blank

More advanced:

- Tracing execution
- Tracing values through code flow (e.g. what is the sequence of values that `x` takes on?)
- Reverse execution (find input that gives an output)
- Minimal fix (given broken code, make it work)
- Theme and variations (working code, adapt to other type of situation/problem)
- Refactoring

More conceptual:

- Draw a diagram
- Label diagram
- Matching problem: two sets of Q/A, match them.

Thinking through the learning taxonomies also helps to come up with diverse types of exercises:

- Bloom's taxonomy: hierarchical skill levels (can you help students to "grow a level"?):
  - Remembering
  - Understanding
  - Applying
  - Analyzing
  - Evaluating

  - Creating
- Fink's taxonomy: complementary types instead of hierarchical:
  - Foundational knowledge
  - Applications
  - Integration
  - Human dimension
  - Caring
  - Learning how to learn

CHAPTER

THIRTEEN

LESSON REVIEW

This presents a checklist for reviewing lessons that already exist. You should also read *lesson-design.md* as well - this is roughly a checklist to the things there.

Remember to keep the *story* of the lesson in mind. Many people are focusing on the small matters (during every change), but only occasionally do people look at the big pictures. That's why a proper review starts with looking at the big picture, instead of adjusting small things and possibly derailing the story.

This is roughly sorted from highest priority for short review to lowest priority for big refactorings.

13.1 Issues

- Look through the issue tracker to see what is relevant, remember and follow up when going through the sections below.

13.2 Lesson guides

Instructor's guide:

- What sections should be taught for what audiences?
- Common pitfalls when teaching
- Any required setup in advance?
  - Any special config files that need to be cleared on instructor's computer when teaching?

Maintainer's guide:

- Learning objectives (necessary to know its place)
- Learner personas (necessary to know its place)
- After you're done analyzing, is there anything in the maintainer's guide you need to update? (The maintainer's guide is probably in most cases the same as the instructor's guide)
  - Design philosophy, how to modify while preserving the overall character.

Student reference guide:

- Anything to fix or already?
- Keep this in mind when you get to episode details.

13.3 Lesson overview

- Is the introduction intrinsically motivating enough? Does it promote an emotional connection to existing problems?
- Student's guide and framing: will a student know when this is relevant to them and how it will benefit them?
  - Doesn't need to include word-for-word learner personas, but should convey this somehow.
- Are the difficulty and prerequisites stated?

13.4 Episode overview

- Read the intro and conclusion to every section/episode.
  - Do they make sense when you read them in order, without reading the text in between?
  - Do they motivate each section well enough (not just explain what, but why it's cool?)
- Do they have learning objectives at top and food for thought at the bottom?
- Are optional or advanced episodes marked as such?
- Does the episode (or lesson overall) say what is next, to keep people interested in growth?

13.5 Episode details

- Read through each exercise (with no other text in between). Does it make a logical progression?
- Exercises labeled with difficulty, optional, etc.
- Optional advanced exercises or material in places where advanced users may get far ahead.
- Each exercise is self-contained: a helper can read just the exercise area and get an idea of what is supposed to happen and why.
- Update the student's reference guide as you are going through the details.
- Remove duplicate or unnecessary information when possible. Things are always added, rarely removed. Shorter is usually better. If something shouldn't be removed, perhaps mark it as advanced or optional.

13.6 Major Refactorings

Always start with the big picture: does it make sense? When refactoring, always start off with backwards lesson design again (see lesson-design.md) and fully go through that.

After the above, do the details. Remember the guides still.

Before you start major refactoring and rewriting, think if it makes sense. Have you figured out why it's the way it is based on the instructor's guide? If you do a big refactoring, make sure you update the maintainer's guide!

Before you embark on a big refactoring step, please pitch your idea in a GitHub issue and collect feedback from others. Maybe even hold a brainstorming session.

CHAPTER

FOURTEEN

WRITING TECHNICAL DOCS

This is a guideline for non-teaching technical documentation, for example HPC infra usage. Since many of us overlap with HPC or other support roles and our CodeRefinery mindset partially overlaps, we have some brief guidelines here.

There is far too much professional information for us to reproduce here, but hopefully this is useful quick reference for a typical person to get started. Check the links at the bottom for more.

14.1 What kind of doc?

- **Tutorial** - emphasis on concepts and mental model, after reading you can do limited things.
- **Reference** - more likely to read if you know the concepts and want to know more advanced stuff.
- **Example** - example of one specific thing. Good for copying and pasting if you know enough to understand it.

"A good tutorial is different from a good reference. Very few things are good at both at the same time." - (I don't remember who, seen in Teaching Tech Together). Thus, both tutorials and reference are useful. Sometimes, we need a tutorial for our stuff and can point to outside material as a reference. People also really want examples they can copy - is your infra similar enough that outside examples can be copied?

14.2 Recommended sections/things to include

(Of course, depending on what type of doc it is)

- Introduction that says *when* this material is relevant and *why* you might want to use it. (*who* it is written for, *when* it was updated). Prerequisites.
- If relevant, there is a *concepts* section at the beginning. Perhaps define a few key terms. Possible prerequisites (required or possibly useful).
- Is the actual *content* good and understandable? Are any conceptual or technical prerequisites mentioned in the introduction (or when they are needed, if minor)?
- *Examples* spread throughout.
- A *conclusion* that wraps up and what happened, why, and what comes next.
- "*See also*" section at the end if relevant.
- If relevant, *exercises* at the end or spread throughout. Even if this isn't the point, it makes people think and makes it minimally useful for informal teaching.

14.3 Style suggestions

(not standards, since of course people do what they want anyway. Not all things apply in all cases)

- **Bold** for definitions or the first time a concept is introduced. Basically, a time where someone is likely to scan up the document to remember what a certain concept is, such as "what's a git stash?"
- *Italics* for local emphasis.
- Never feel bad to use simpler text, in the best case someone can now understand, in the worst case there's less mental effort.
- Use an active/imperative voice (Y does X, do X to get Y), not passive (X can be done by Y). Try to use present tense.
- Make the docs skimmable - by looking at headings and first words in each section, can you figure out what you need to focus on?
- Use gender neutral text, of course.

14.4 Style guide

(Is it worth making a minimal academic HPC/tech doc style guide? The benefit would be that we can share stuff better. There's no need to emulate or reproduce actual professional ones, since we probably aren't that formal.)

14.5 Other ideas

Consider documentation-driven development. Write basic docs, review, then implement it.

14.6 See also

Of course, there is far more professional information than you can find above.

- The Write the Docs guide seems useful - especially about organizational aspects.
  - Principles
  - Styles and list of style guides, though most are probably too long for hobbyist reference.
- *lesson-design.md* might be useful to understand. The "backwards lesson design process" is especially important to think about: while you don't have exercises, you do have goals to accomplish you can design to.

Download this guide as single-page HTML, pdf, or epub.