Group Handbook

A guide to help you start your scientific adventures

Version 1.4
4/21/16
hackingmaterials.lbl.gov
maintained by Anubhav Jain

Table of Contents

Preamble

"Organization is a means of multiplying the strength of an individual".
- Peter Drucker

Welcome to the HackingMaterials research group! The purpose of this guide is to ease your transition into our research group and help make your time here as productive and enjoyable as possible.

An editable copy of this guide, where *everyone* is welcome to make comments and suggestions, is always available at:
https://bit.ly/2huxUJW

You can also download the latest "PDF release" of this guide at
https://hackingmaterials.lbl.gov/handbook.pdf

The page size of this document is A5, which makes it work well for **two-page** viewing and printing. We suggest you give that a try.

We thank Valve software (a video game company) for openly publishing their employee handbook, which helped inspire this effort. Also, thank you to those who help create and update this guide; each of us gains from the contributions of those before us, and we hope you are able to pay it forward to future members by contributing as well.

About our group

Our group is located at Lawrence Berkeley National Laboratory (LBNL) in Berkeley, California. LBNL is managed by the University of California at Berkeley, which is located just down the hill. 13 Nobel prizes have been awarded to scientists from LBNL. The lab has spectacular views of San Francisco, which is located across the bay and is about a 30 minute drive or BART train ride away. Berkeley itself is a vibrant city of 115,000 people filled with cafes, restaurants of all types, and cultural activities.

Our group aims to tackle some the most important problems lying at the intersection of materials science and computer science. We differ from a traditional materials theory group in our emphasis on building long-term software, in leveraging large supercomputers, and in applying statistical learning to materials problems. Most of our projects use a "materials genomics" approach, a new mode of research that has tremendous potential to discover new materials and to improve our fundamental understanding of how materials behave.

Our research group is its own microcosm within the materials science theory efforts at Berkeley. In the same way that Berkeley is a small city adjacent to the bigger city of San Francisco, our group is a smaller unit

linked to the larger theory groups of Kristin Persson, Gerbrand Ceder, Jeffrey Neaton, and Mark Asta – creating a close-knit community of materials theory within Berkeley. We also collaborate with groups external to the Berkeley area, and thus it is almost always the case that *someone* within our collaboration circle has experience with any new methods or applications you might be interested in. We hope you are able to leverage many of these resources during your stay!

Many new discoveries remain to be uncovered in the field of materials design and in our relatively new subfield of materials informatics. Your contributions are urgently needed to make this new vision a reality - welcome!

Before you arrive

Although many things can only be taken care of after arriving at LBNL, here are a few simple things you should do in advance.

Join the Slack group!

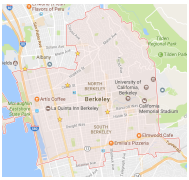
The berkeleytheory Slack group spans several research groups at LBNL. There are channels to ask housing questions, programming questions, science discussion, and general hijinx.

How? **Contact Anubhav - he'll introduce you.**

Order a computer!

Postdocs and graduate students - let's order your workstation in advance so that it's ready by the time you arrive.

How? **See Appendix on purchasing a computer.**



Find a place to live!

Berkeley and the surrounding areas are wonderful places to live, but finding an apartment can be difficult and is best done with knowledge of the various neighborhoods.

How? **See Appendix on housing.**

After arriving at LBNL

Welcome to Berkeley! Here are a few pointers for getting started.

Getting set up to work

Here is a checklist of things you should complete within your first week:

- ❑ **Set up your computer**. See Appendix documentation on how to do this and some recommended software to install.
- ❑ **Complete the checklist that HR gives you in your welcome package.** Note: postdocs cannot select retirement plans; an obligatory UCRS DCP is deducted from your pre-tax salary each month but the money belongs to you.
- ❑ **Complete all LBNL training courses.**
- ❑ **Request to be added to the two email lists for the group.** Alireza can add you to *hackmat@lists.lbl.gov* and *hackmat-noboss@lists.lbl.gov*. The latter does not include Anubhav so you can speak freely!
- ❑ **Set up the employee wifi**. Note that the visitor wifi is open access. To connect to employee wifi, go to ***https://software.lbl.gov/***, search for "Wireless Networking", and download the configuration file.
- ❑ **Request access to the group's Google Drive folder.**
- ❑ **Install VPN for connecting to the lab network from home.** For example, this lets you download research articles from home. See ***https://software.lbl.gov*** for instructions on installation.
- ❑ **Obtain user accounts for any computing resources you may be using**. See documentation later in this handbook.

- ❑ **Schedule a 10-minute weekly meeting time with Anubhav.** Ask about any other meetings.
- ❑ **Obtain a license for any software packages you might be using.** For example, you may need to be added to the VASP users list (for VASP, you should also register for the forum.)
- ❑ **Set up the printer.** It should appear on your network as "Toshiba e-studio" and you should be able to connect if you are on the LBNL network (e.g., if you are using an ethernet cable). The printer is located in the corner of the third floor of building 62 near the water fountain.
- ❑ **Request after-hours access**. By default, you will not have off-hours site access to building 62, *i.e.*, on weekends, holidays, and from ~6pm to 7am on weekdays. To obtain off-hours site access, email the Site access office at the lab (*siteaccess@lbl.gov*) and cc Anubhav.
- ❑ **Ask to read the proposal that funds your work.** This will help explain the impact of your project, the long-term plans and goals, and how your project fits in with other efforts.
- ❑ **Have your picture taken for the group web site.** Coordinate this with Anubhav, who will take the picture.
- ❑ **Update your name tag outside your office.**
- ❑ **Add your name to the 62-253 mail room.**
- ❑ **Say "hi" to your neighbors!** Working here will be more pleasant if you get to know some of the people around you. One good time to introduce yourself is when you see people eating lunch in the kitchen area.

Getting situated in your office

There is no rule that says your office must be dull and generic. Decorate your lab space with photos, posters, or other personal touches. You are likely going to be sitting in this office for quite some time so you should take a moment to make this space your own.

Food and coffee

There is a common kitchen in building 62 with microwave, fridge, and coffee machine. I don't know who the coffee machine belongs to but you can try to make friends with whomever that is. There is a coffee machine in the 2nd floor of Molecular Foundry ($1/cup or $15/month) that some of the other postdocs in the group can tell you more details about. For *emergency* coffee situations, Anubhav has an espresso machine in his office that you can borrow (but wait for the go-ahead to come in, since he is often in the middle of a videoconference).

There are no vending machines in building 62, but if you go down to the first floor you walk to building 66 and there are some vending machines on the bottom floor. There are also some vending machines on the bottom floor of 67 (Molecular Foundry).

The only real food is in the lab cafe, which is about a 15 minute walk from our building. The menu is posted at (some items change weekly): ***http://www.bayviewcafelbl.com***

Mail and fax

The incoming and outgoing mail corridor is located in building 62, second floor. You can find an empty slot and put your name there; sometimes, your mail will end up there, other times your mail will just end up in a common pile, so you may need to check both. Please do not have any personal (non-business related) mail sent to your LBNL address - this is not allowed.

If you would like to mail something internally (including the benefit office, which is not on the hill), first get an envelope either from 62-309 or from building 66 (room 237-250, make a left right after entering). Scratch off all the previous mail stops and write down the destination mail stop (something like: 90P-0101) and then put it in the outgoing mailbox in the mail corridor in 62. There are also some miscellaneous mailing supplies in 62-309 on the shelves.

There is a fax machine in 62-309. For outside numbers, dial 9 first, then the country code (1), then the rest of the number.

Equipment and conference rooms

Should you need it, Anubhav has a projector that you can borrow. If he is not in the office, feel free to go in and just grab it (it is visible on the gray shelf). Just leave a note and remember to return it.

Anyone can reserve conference rooms through the LBNL Google calendar. Make sure you are logged in to your lbl.gov account.To see the availability of a room, just add the conference room calendar to your list of calendars (find the area that says “Other calendars” and then type the

room into "Add coworker's calendar"). To book a room, add an event to your own Google Calendar and use the option within Google calendar to add a room. You will see a list of LBNL rooms displayed. *e.g.*, 62-203 (big main room) and 62-253 (smaller room with poor wi-fi). Note that when adding a room, you will see a number in parenthesis like (20) - that is approximately the number of people that the room can accommodate. For more detailed information, see ***https://commons.lbl.gov/display/fac/Conference+Rooms***.

The Panic Monster

The Panic Monster is a red doll in Anubhav's office based on a blog post from *Wait But Why*. If you see the Panic Monster on Anubhav's desk, it is best not to bother him. If you see the Panic Monster on *your* desk, it means you have lots of work to catch up on and you need to get working!

Postdoc union

Note that Berkeley postdocs have unionized with the *International Union, United Automobile, Aerospace and Agricultural Implement Workers of America* to obtain collective bargaining agreements. Joining the union is an option, and many of the details are present here: ***http://uaw5810.org*** You can also message the Slack group to get opinions from the current postdocs in the group.

Vacation days

You will receive a set number of vacation / personal time off (PTO) days that will be outlined in your hiring package. For union postdocs, the union has currently negotiated 24 PTO days per year along with other benefits.

You should coordinate the specific days of vacation and personal time off with Anubhav, especially for an extended absence.

What to do if you're sick

If you're sick, ***do not come to the office***. This is very important; otherwise, you can get others sick and potentially bring down the productivity of the entire group. Instead, work from home or take a sick day to rest, relax, and recover. Simply e-mail Anubhav and let him know what you're doing. Just don't come into the office!

Filling out your timecard (LETS)

Every month, you are required to fill out your timecard at ***https://lets.lbl.gov***. This is mostly straightforward but here are a few pointers:

- To fill out your work hours, leave "Earning Type" as Regular and "Shift" as 1. For project id, activity id, and days, use the information Anubhav gives you. You do not need to fill out the work/job number or the specific days on which you worked.
- For sick or vacation days, set "Earnings Type" to the appropriate value. You will need to enter both the number of days as well as the specific dates that you took sick leave/vacation.

- In total, the number of days should match the "Work days" listed in the top-left. If you took vacation or sick days, simply deduct those number of days from your project (proportionally if you have multiple projects).
- When finished, click "Run Report", then "Release".
- Note that your "Leave Balance" in the bottom-left assumes 8 hours per day.

Other issues

If you are struggling with stress or other personal problems, you can contact the LBNL Employee Assistance Program (EAP), which provides free and confidential counseling, consultation, and referral for LBNL staff. If you are comfortable doing so, you can also discuss the problem with Anubhav to brainstorm if there are ways forward.

Places to work outside of your office

Anubhav is much more interested in your research output than where you work. Indeed, one of the advantages of choosing computational science as a career is that it can afford you some more flexibility than other jobs. Overall, you are encouraged to work where you feel best *from time-to-time* to maximize your energy and productivity. The more accurate policy is that the more productive you are in terms out output (see "questions for self assessment" section), the less Anubhav cares about where you are doing your work.

Some places to work apart from your office include:

- The Molecular Foundry 3rd floor lounge

- The LBNL coffee shop and library reading room (downstairs from the cafeteria cash register area)
- The UC Botanical Garden (free admission for LBNL)
- UC Berkeley campus, including the Free Speech Movement cafe which has rotating newspaper headlines
- Downtown and campus-area coffee shops

The Molecular Foundry 3rd floor lounge has a view overlooking San Francisco.

The library reading room near the lab cafe is a quiet and attractive place to work.

Making purchases

You are encouraged to make purchases that are likely to save you a lot of time. For example, if a commercial version of a software is superior to open-source alternatives, then you should purchase the commercial version. Your time is valuable and if we can solve a problem with funds, then we should try to do so.

Purchases are usually paid for through **project ids** that Anubhav can provide you with. For items less than $100, you should initiate purchases on your own provided that you know the correct project id (just let Anubhav know afterward so he is aware of the charge). For items greater than $100, contact Anubhav first.

The procedure for making purchases depends on the purchase type:

- **Software**: Many popular commercial software libraries (e.g., Microsoft Office) can be purchased through *software.lbl.gov*. At check out, the approver for the purchase is Micah Josh Liedeker.
- **Office supplies, computer accessories**: Check the LBNL's Ebuy (*not* Ebay) first via *procurement.lbl.gov*. If the item or an equivalent is available, this is the easiest way to make the purchase (for both you and LBNL administration). At check out, the approver for the purchase is Micah Josh Liedeker. Otherwise, see below.
- **Books**: First, see "Resources for learning new topics". After that, if you'd still like to purchase a book, first check if the book is available on Ebuy - this is the simplest purchase option. Otherwise, follow the instructions below for "other purchases".

- **Other purchases:** You should coordinate other purchases with the division's administrative contacts (currently esdradmin@lbl.gov). Most likely, someone will ask you to fill out a simple purchase order.

Booking conference travel

Graduate students and postdocs are encouraged to attend conferences for professional development and to broadcast your work to the research community. Many if not most people learn about new research by hearing about it at a conference. Thus, if you want people to know about your work, you must be willing to tell people about it. Anubhav encourages you to be proactive in seeking out conferences that may be beneficial for you to participate in.

You should identify conferences you'd like to attend several months (usually ~4 months, perhaps ~6 months for international travel) in advance. Usually, this is around the same time that abstract deadlines are due.

Once you have identified a conference you'd like to attend, please take the following actions:

- Tell Anubhav about the conference and what project you'd like to present
- Work with Anubhav to submit an abstract
- As soon as possible - submit a conference travel request form. This form is a very basic (i.e., 2 minutes to fill out) Google spreadsheet: ***http://bit.ly/2niepv7***
- If you do not submit the travel request form several months in advance, you may not receive LBNL approval to attend.

- If you haven't done so already, make sure your travel profile (*e.g.,* your frequent flier programs) are completed for the lab. E-mail Charlotte Standish if you don't have one yet.

Once you have received approval to attend the conference, please take the following steps:

- Make sure you register for the conference in time to receive any early registration discount *(normally on one's own credit card then reimbursed later)*
- Book a hotel *(normally on one's own credit card then reimbursed later)*
- Book a flight - please do this early to avoid last-minute flight rate spikes *(normally booked in coordination with Charlotte Standish with LBNL making the booking. This works better if you identify desired flights in advance, otherwise give Charlotte the preferred dates and times.)*
- If you are planning to combine vacation and travel, remember the lab's policy of taking only one vacation day per two work days. Note that days spent traveling to and from the conference count as work days.

In terms of travel receipts and reimbursement:

- If you are traveling with funding through LBNL (i.e., most cases), you do ***not*** need to save receipts for meals. You will receive a per diem instead. You also do not need receipts for taxi rides under $75, although Anubhav usually submits them anyway when he has them. You also do not need to save your actual airplane tickets for lab-purchased airfare, although again Anubhav usually submits these anyway.

- If you are traveling with outside funding (e.g., the conference organizers are going to reimburse you), save all receipts and tickets as they may be needed for reimbursement.
- When returning from travel, coordinate reimbursements with Charlotte Standish. You can also ask Charlotte about any travel questions that might not be covered here.

Pro tip: If you want to see the status of your conference requests, log in to this sheet with your LBNL account: ***http://bit.ly/2n6XCe3***
You can filter the sheet to your requests by right-clicking on the name column and choosing the filter option.

Asking your advisor for research help

Face-to-face meetings: weekly 10-minute checkups and targeted meetings

Anubhav will schedule a time to check up with you every week for 10 minutes. The length of these meetings is intentionally short and you do not need to present slides or prepare any formal presentation. Some of the things you can do during these checkups:

- mention anything that is impeding your progress (e.g., lack of equipment, lack of response from a collaborator, long queue wait times at supercomputer, etc.)
- report what you worked on the last week and present your goals for the next week, month, or 3 months to confirm confirm that you are on the right track, not repeating previous work, etc.

- introduce a major problem you are facing and that requires a longer, targeted meeting to brainstorm a solution (just present the problem in enough detail)
- solve very small problems, such as getting feedback on 3-4 presentation slides
- request a decision about something
- pitch a Friday Afternoon Tinkering (see later in this handbook)

Longer, targeted meetings are welcome so long as you have a clear purpose for them. In particular, if a difficult decision needs to be made or you'd like to brainstorm a technical problem, a longer meeting will almost certainly work better than e-mail. To schedule a targeted meeting, you should first e-mail me and tell me why you'd like to meet (or simply describe your need at a 10-minute meeting). If a meeting is in fact the best solution, we will work out a time and place to have the meeting. You can use Anubhav's public calendar to suggest a few possible times. The best way is to add his calendar to yours through LBNL Google Calendar (see instructions in the section about Booking Conference Rooms). Another way is to access his calendar through this public link:

http://bit.ly/2ncHcPo

Note: Please do not simply "drop by" Anubhav's office to ask questions unless the matter is urgent or if the issue is of a more personal nature (in which case, please do not hesitate to drop by).

Email help (and general guidance)

From time to time, you will encounter problems, require suggestions, or otherwise need assistance from your advisor. This is normal, and asking for help is encouraged so long as you have done your best to solve the problem yourself. Sadly, it is all too easy these days to send e-mails without first investigating a problem yourself, and it is important to remember that your advisor gets many dozens of emails per day. Thus, if the question is important and difficult enough to ask your advisor, you should take the time to address the following four questions[1]:

❑ **What is the problem?**

Clearly describe the problem, *starting from the beginning.*

❑ **What is the CAUSE of the problem?**

For example, your immediate problem may be that you need more computing time. But the *cause* of your problem is perhaps that you want to determine the best ordering of a disordered compound and that you anticipate that this will require running many calculations.

❑ **What are all the possible solutions to the problem?**

List *all* possible solutions that you can think of, including those that you may not know how to implement or think might not work. If you don't have a solution, list all the avenues you tried (e.g., Google search terms) to find one. If you already tried some solutions but they failed, summarize that information here.

---

[1] These guidelines are adapted from Dale Carnegie.

- ❑ **What solution do you suggest?**

Provide your reason for suggesting this solution

More often than not, taking the time to answer these questions leads to you solving your own problem. In the cases where that is not true, these responses will make brainstorming solutions to your problem more effective and will also allow Anubhav to provide feedback into your process of generating all possible solutions.

Software help groups

If you have problems with software, and in particular the software maintained by our group and our collaborators, you should contact the appropriate help group. The documentation for the software will list what that channel is; if not, try the Github Issues page. If you contact Anubhav, make sure you address the four questions above as well as provide everything needed (files, test code, etc) to quickly reproduce and debug the problem.

Two other ways to get software help that are more self-guided are:

- If you are having trouble using a particular class or function, look for unit tests within the code, which often demonstrate how to use the class or function
- If the class or function has a unique name (e.g., MaterialsProjectCompatibility), another option is to both Google *and* search on github.com for the particular class/function. The github.com search will often reveal code snippets from users all around the world.

Friday Afternoon Tinkerings (FATs)

"You'll learn infinitely better and easier and more completely by picking a problem for yourself that you find interesting to fiddle around with, some kind of thing that you heard that you don't understand, or you want to analyze further, or want to do some kind of trick with - that's the best way to learn something".
- Richard Feynman

"...it is certainly all right and potentially very productive just to mess around. Quick uncontrolled experiments are very productive. They are performed just to see if you can make something interesting happen".
- E.O. Wilson

Long-term members of the group (*i.e.*, graduate students and postdocs) are encouraged to spend Friday afternoons on a project of their choosing. This is similar to the "20% time" afforded by companies like Google. You can pick any project you think is important or interesting, as long as it is related to our group and somewhat related to your main project theme. It is best if the project is scientific in nature, but it can also be software related (*e.g.*, to try a new visualization library or to improve the scaling of calculations at supercomputing centers). The most important aspect is that you should be able to get a result (whether positive or negative) in a short time (see "metrics for successful FATs").

You are the boss for your Friday Afternoon Tinkerings (FATs).

FAT rules

- To initiate a Friday afternoon tinkering, simply email Anubhav to let him know that you are going to try this. This will also

serve as a reminder to him to leave you alone on Friday afternoons.

- At the end of every month that you are doing a FAT, you will be expected to provide 3 slides that describe your project and its status.
- Based on the slides, Anubhav will suggest what to do next. Regardless of Anubhav's suggestions, you are allowed to continue for another month (2 months total) on the same project theme.
- At the end of 2 months, Anubhav will either (i) tell you to stop working on this particular project, (ii) ask you to continue hammering, or (iii) make your project more official and extend your time on it.
- One Friday afternoon tinkering at a time - pick the most important or most exciting one if you have several ideas.

Metrics for successful FATs

The main metric for success of a FAT is whether you were able to prototype and test an idea. It does not matter as much whether the outcome was positive or negative, so long as you were able to obtain a clear result.

For example, let's say your project is to represent crystal structures as graphs. Over 4 afternoons, you do basic research and develop some rudimentary code to represent crystals as graphs. You include these graphs as descriptors for predicting a several materials properties, discover the results are not so good compared to existing descriptors, and conclude that the idea probably doesn't work very well after all. This

is a ***successful*** project, despite the negative outcome, because you performed a real experiment and got a result.

As a counterexample, let's say you are interested in developing a predictive theory for metastable states. You spend the first 4 afternoons doing literature research and talking to colleagues about how to proceed, spending most of your time brainstorming. You spend 4 more afternoons and you think you might have sketched out a good plan of how to tackle this problem. This is an ***unsuccessful*** project because you did not actually build anything or test any ideas. Even if you felt this was productive research, this was not a FAT project.

Main message: you must prototype, test, and gather data during your FATs!

Our computing systems

Our group's main computing resources are:

- NERSC (the LBNL supercomputing center, one of the biggest in the world)
- Lawrencium (starting sometime in 2017, our group will own 4 high-performance dedicated nodes on this cluster)
- Argonne Leadership Computing Facility (sometimes)
- Oak Ridge Leadership Computing Facility (sometimes)

At any time, if you feel you are computing-limited, please contact Anubhav so he can work with you on finding solutions.

NERSC

To get started with calculations at NERSC:

1. Ask Anubhav about whether you will be running at NERSC and, if so, under what account / repository to charge.
2. Request a NERSC account through the NERSC homepage (Google "NERSC account request").
3. Someone at NERSC will validate your account and assign you computing hours
4. At this point, you should be able to log in, check CPU-hour balances, etc. through "NERSC NIM" and "My NERSC" portals
5. In order to log in and run jobs on the various machines at NERSC, review the NERSC documentation
6. In order to load and submit scripts for various codes (VASP, ABINIT, Quantum Espresso), NERSC has lots of information to help. Try Google, e.g. "NERSC VASP".
    a. Note that for commercial codes such as VASP, there is an online form that allows you to enter your VASP license, which NERSC will confirm and then allow you access to.

Lawrencium

Lawrencium is somewhat different than NERSC in that we must maintain our own software environment and pre-installed binaries for common codes are not available. Thus, maintaining the software environment at Lawrencium is a group endeavour. Our negotiation to purchase nodes on Lawrencium is currently in progress. This handbook will be updated when we have completed the purchase and it is ready for the group's use. Note that if our purchased nodes is not enough to

sustain our computing needs, it is also possible to pay per CPU-hour on Lawrencium as well as to increase our purchase order. Ask Anubhav if you think you need this.

ALCF and OLCF

Both ALCF and OLCF are "leadership computing facilities" meaning that they operate some of the fastest computers in the world. The strength of these facilities is that they offer very large amounts of computer time available for users; the weakness is that is much more difficult to use these computers. Therefore, it is generally only worth using these resources if you have a significant amount of computing to do (*i.e.*, at least 1 million CPU-hours). Contact Anubhav if you think an account on ALCF or OLCF would be useful.

Our software stack

A brief summary of our software stack includes:

- **pymatgen / pymatgen-db** - for representing and analyzing crystal structures, as well as setting up/performing manual calculations
- **FireWorks** - for executing and managing calculation workflows at supercomputing centers
- **custodian** - instead of directly running an executable like VASP, one can wrap the executable in custodian to detect and fix errors
- **atomate** - for quickly defining multiple types of materials science workflows
- **matminer** - for large data analysis and visualization

We also heavily use the **Materials Project** database.

To learn how to use the software stack, you can consult the documentation of the individual codebases as well as review the following resources:

- The 2016 Materials Project workshop (note that MatMethods is now called atomate):
  https://github.com/materialsproject/workshop-2016
- The 2014 Materials Virtual Lab presentations:
  https://materialsvirtuallab.org/software/
- The Materials Project YouTube tutorials:
  https://www.youtube.com/user/MaterialsProject

Resources for learning new topics

Slack

If you have a specific question, sometimes the easiest solution is to post it to the Slack group and crowdsource the answer.

Books: LBNL, UC Berkeley, public libraries, and the "group library"

As an LBNL employee, you can get access to almost any book you'd like using various channels:

- LBNL has its own library, but it is small and unlikely to contain the book you want.
- LBNL employees can borrow books from the UC Berkeley Library collection using your LBNL ID. You can even reserve

the book online and have it delivered to the LBNL library office in building 50, saving you a trip down to campus (this is what I do). Log in through ***http://oskicat.berkeley.edu*** via "My Oskicat" and choose the LBNL login option.

- Your local library (e.g., Berkeley Public Library) often participates in Interlibrary loans. For example, the "Link+" system at Berkeley Public Library connects to many other university libraries in the area. Again, you can have the items delivered to your local library. This service is extremely useful when an item cannot be found at UC Berkeley or if that item has a long waiting list.
- Our group has some technical books that you can loan. See ***Appendix I: Group library***.

You can also purchase books with research funds.

Materials Science

"Don't despair of standard dull textbooks. Just close the book once in awhile and think what they just said in your own terms as a revelation of the spirit and wonder of nature".
- Richard Feynman

It can be difficult to find resources that explain concepts in materials science clearly. Often, struggling through multiple attempts to understand a topic using several different resources in a patchwork and non-linear fashion is the only way forward. That said, the resources listed below are particularly helpful.

Density functional theory

For beginners to density functional theory, I would recommend the book "***Density Functional Theory: A Practical Introduction***", which truly achieves what it states by providing physical insights and relevant information rather than just list equations. A copy is available within the group.

If you are interested to explore applications of density functional theory, you might try the E-book from Professor John Kitchin:
https://github.com/jkitchin/dft-book
Note that this book has chosen to use the Atomic Simulation Environment (ASE) to set up simulations rather than the pymatgen code that we prefer, but that is a minor point.

Finally, for specific calculations with VASP, there are resources online from a 2016 workshop conducted at LBNL, including videos and training materials:
http://www.nersc.gov/users/training/events/3-day-vasp-workshop/
http://cms.mpi.univie.ac.at/wiki/index.php/NERSC_Berkeley_2016

General materials science topics

To gain a quick introduction to many topics in materials science, you might try the (horribly-named) web site from the University of Cambridge: Dissemination of IT for the Promotion of Materials Science (DoITPoMS):
https://www.doitpoms.ac.uk
The explanations in this site are very basic, but what they do cover is well-explained incorporates helpful visuals. Although you won't ever

master a topic from this site, it is often a good starting point that can help you unlock a more intermediate resource.

There are also some nice chapters in the following e-book:
https://en.wikibooks.org/wiki/Introduction_to_Inorganic_Chemistry
For example, Chapter 5 has a nice rundown of common crystal structures.

Online tools

A nice tool for visualizing phonon modes is:
http://henriquemiranda.github.io/phononwebsite/phonon.html

Computer programming

Note that there are usually many excellent resources to choose from when learning computer science topics. You usually have the flexibility of choosing to learn from a book, a video series, or even interactive tutorials like ***www.learnpython.org***. Use the list below as potential starting points, but there exist many other high-quality alternatives you can find on your own and may be even better-suited to your needs.

Python

For beginners to Python, you might try the book "***Head First Python***". It is a fun and easy introduction to Python. For advanced programmers, you might try "***Expert Python Programming***".

Data mining and Data Analysis

For learning basic data mining libraries (*pandas*, *scikit-learn*) as well as some skills like using *git* and *Github*, you might try the online YouTube

videos from Kevin Markham, an educator at Data School. These videos also do a good job of pointing you to supplementary material:
https://www.youtube.com/user/dataschool
https://github.com/justmarkham

You might also try the book "***Python for Data Science For Dummies***" (please note: this is different than "*Data Science for Dummies*").

For a more materials-centric view, you can try working your way through the *Machine Learning In Materials tutorial* in the Appendix of this handbook.

A very comprehensive set of suggestions for further resources is listed here:
http://bit.ly/2jHXIVJ

MongoDb

A (now somewhat old, but still clear) resource for beginning to use MongoDb is the "*The Little MongoDB Book*":
https://github.com/karlseguin/the-little-mongodb-book

There is also an extensive library of webinars on MongoDb on their official web site.

Ten questions for self-assessment

You might be curious as to whether you are on the right track from a professional standpoint. You can ask Anubhav to give you feedback periodically, and you should do this at least every 6 months or so. Here is

a cheat sheet of ten things he considers when thinking about your progress.

1. How self-driven is your work?
    a. I am far ahead of my supervisor in understanding and guiding my project, so it's necessary that I conceive/design/imagine/build most of what I do. i.e., I am given a vague topic to work on by my supervisor and it's my job to determine both the important problems and design the solutions.
    b. I originate maybe 50% of the ideas that I work on; the other 50% are from my supervisor. Or, given a good description of the problem by my supervisor, I figure out the solution largely independently.
    c. Almost all of what I do was sketched out by my advisor, including the problem and the rough solution; my job is to implement those ideas.
2. When I am assigned a task, I usually complete it:
    a. To an even higher quality standard than asked for and/or much quicker than expected
    b. approximately on time and well-tested and robust so that I know that my solution works under diverse situations. I can declare "mission accomplished successfully" the vast majority of the time.
    c. To minimally achieve the original goal, thus often requiring future revision; OR very late; OR usually by getting someone else to solve most of the hard parts for me
3. Compared to others in the group, I:

    a. help them more than they help me
    b. help them about the same as they help me
    c. help them less than they help me
4. Regarding the relevant scientific literature for my project, I:
    a. regularly impress my supervisor by integrating new and important papers into my research that were not on my supervisor's radar
    b. have about an equal share of papers I receive from my supervisor versus papers I have discovered on my own (and subsequently adapted my research to account for those papers, i.e., have read and understood them)
    c. typically am the recipient of interesting papers to read from my supervisor
5. Regarding independent Friday Afternoon Tinkering projects, I have completed:
    a. At least one every 3 months
    b. About one every 6 months
    c. None
6. In the last 9 months, I have submitted to a journal as first author:
    a. Multiple papers, or one stellar paper
    b. One pretty good paper
    c. No papers
7. When I present a draft of a paper and/or presentation to my supervisor, usually I get back:
    a. Minor revisions
    b. Medium level of revision
    c. Major revisions

8. Will my main work serve as a lasting contribution that others will use and refer to in 5-10 years time?
    a. There is a good chance my work will remain important even after 10 years
    b. Probably 5 years, 10 years is a stretch
    c. Honestly, probably not
9. When my supervisor assigns tasks (e.g., at in-person meetings, etc.), I:
    a. complete tasks quickly and efficiently and provide my supervisor with an update
    b. get around to doing almost all tasks eventually
    c. often forget to complete tasks (e.g., forgot to write it down, etc.) and often need to be asked a second time
10. How is your passion and enthusiasm level?
    a. I feel extremely excited and happy about work
    b. About normal
    c. I feel burnt out or demotivated

If your answers are mainly (c), the questionnaire is probably telling you something that you already know -that you should take some time to reflect on your situation. You might also schedule a meeting with Anubhav to discuss things. If you are answering mainly (b), then you are likely doing fine but it may be worth brainstorming if it's possible to move into category (a) for one or more of your responses. Otherwise, continue the great work!

Fun things to do in the area

Make time to explore some of its recreational activities in the Bay Area. Although there are probably hundreds of online and print resources that

can help guide you to things to do, here are a few select ones to start you off:

- ❑ UC Botanical Garden (walkable from our office and free for LBNL employees)
- ❑ Berkeley Marina (walking) - either the boardwalk or the Cesar Chavez loop
- ❑ Ohlone Parkway Trail (easy bike)
- ❑ Indian Rock park
- ❑ "Off the Grid" food trucks
- ❑ Explore the Elmwood shopping area
- ❑ Detour App - guided tours for locals through your phone
- ❑ Berkeley Jazz / Theater
- ❑ UC 50% off performing arts at Zellerbach Hall
- ❑ Tilden State Park, e.g., Lake Anza trail
- ❑ Bike the Golden Gate bridge to Marin (longer bike)
- ❑ SF Film Fest
- ❑ Baker beach walk up to Golden Gate Bridge
- ❑ Ice Cream - Mitchell's, Humphrey Slocombe, Bi-rite, Ice Cream Bar (skip the line, go directly to the back bar and order a "New Orleans Hangover" - non-alcoholic)
- ❑ Muir woods
- ❑ Drive up to Mount Diablo
- ❑ Hike - Stinson Beach / Matt Davis trail
- ❑ Muir Beach Lookout
- ❑ Wine country / Sonoma
- ❑ Point Reyes Lighthouse - Elephant Seal season Dec - Feb
- ❑ Route 1, Big Sur, 18-mile drive. Would very much suggest going down to Bixby bridge area at least once.
- ❑ Explore Monterey and Carmel-by-the-Sea

- ❑ Lake Tahoe - skiing in the winter, hiking/biking/cruises/tourism/casinos in summer

Appendix A: Finding a place to live

Note: It is encouraged that readers of this document also contribute to it when they have gained experience with their own housing situation by adding comments or emailing Anubhav with their suggestions.

Resources for finding housing

Unless you are part of a program that assists you with finding housing, you must find a place to live on your own. Some resources for finding housing include:

- ***http://www.zillow.com/***
- ***https://calrentals.housing.berkeley.edu/***
- ***http://sfbay.craigslist.org/search/eby/apa?***
- ***http://csee.lbl.gov/Housing/Other_Housing_Resources.html***

You can join the LBNL postdoc mailing list (***http://bit.ly/2nAHAXE***). You do not have to be a postdoc to join. Keep an eye on the posts for room/apartment to rent as well as moving sales.

Notes on the Bay Area housing situation

The Bay Area is a very nice place to live, which has the consequence of many people wanting housing here. Thus, one of the few problems with this area is the very high price and competition for housing. Some things you should be aware of:

- prices in the $2000/month range for a very basic apartment are normal - and can easily go up from there.

- buildings tend to be older, and amenities like dishwashers and heating/cooling are hard to find.
- it is normal to have a lot of competition for a place such that you must agree to a lease on the spot or risk losing out. Examples:
    - Anubhav thought he had finalized a place to live for his first year in the North Berkeley area, but during the signing period another bidder put in an offer for $300/month greater and he thus lost the place.
    - One of Anubhav's postdocs thought he had finalized a deal for a place to rent but was late to an appointment to meet with the owner and, even though he texted about the situation somewhat in advance, ended up losing the offer on the spot for this single mistake.

You might not expect these kinds of situations unless you are from a similar area like NYC, so please be aware of them.

Commuting

Note that if you use public transportation daily, you should consider signing up for LBL's program which lets you deduct a bus or BART pass as a pre-tax expense. See ***http://www.wageworks.com/*** for more info.

Biking here is common and there are many bike lanes and shared car/bike routes, but you still need to be careful as biking to the lab will mean going through traffic. The LBNL shuttle has bike racks so you can bring your bike up to the lab with you on the shuttle rather than bike uphill.

Note that the Nextbus app and website will give times that the LBL shuttle (and also city buses) are anticipated to arrive at various stops.

If you are in a rush or just need to get around town, Uber and Lyft are apps that can help get your a ride; the fees tend to be pretty low, especially with UberPool if you're not in a hurry.

General suggestions when evaluating a place to live

- Look for the nearest grocery store
- Look for the nearest pharmacy
- Do a search for restaurants. Often, the density of restaurants in a place will tell you whether there are other things there as well.
- Perhaps do a Google Street View walk-through of the neighborhood
- Do a Google Transit search on how to get to the lab. Note that to get to the lab itself, you cannot take public transportation. Instead, there is a lab shuttle from several spots in downtown Berkeley and near campus, so you might want to gauge how to get to the nearest shuttle stop. Google "LBL shuttle map" to see the locations of the stops.
- Remember that Uber is very convenient in the Berkeley area, so not everything needs to be ideal location-wise if you need to just get somewhere once in awhile.

A note about UC Village

Many postdocs, especially those with families, find that UC Village (sponsored housing from UC Berkeley and LBNL) is a nice place to live and also enjoy the community. Anubhav doesn't have any personal experience with UC Village so it is best to research for yourself through a Google search.

What are the different neighborhoods like?

Our group maintains a document that is separate from this handbook about what different neighborhoods are like. Please ask Anubhav for it, and don't forget to ask follow-up questions on Slack!

Good luck!

Appendix B: Purchasing a computer

Most long-term appointments (graduate student, postdoc, staff) will mean purchasing a new computer. **Short-term appointments (e.g., internships) will not involve a computer purchase unless otherwise stated - you will instead receive an excellent computer from the group's stock.**

Mac, Windows, or Linux?

You should buy a Mac, and probably a Macbook. Although this sounds extreme, and may even induce strong feelings if you are used to a different system, in practice this has never been as much of a problem. Note that I am not an Apple fanatic but simply find that these are the

best systems for our type of work because they contain many of the advantages of both Linux and Windows systems in a single package.

Why not Windows? Anubhav used Windows for a very long time; it is nice, but a couple of things make it non-optimal for our work. There is no native Terminal, which you will use heavily, and programs like Cygwin are poor substitutes. Certain seemingly minor decisions made by Windows (directory slashes, line endings) are different than those from Linux, making interoperability between Linux/Mac and Windows systems more problematic (e.g., copying files to and from supercomputing centers requires auto-converting line ending format).

Why not Linux? Linux is fine, but Microsoft Office is not available (which is used by us and most of the materials science research world) and OpenOffice is a poor substitute. Certain videoconferencing software doesn't work well with Linux.

How about Mac? I have my complaints about it, as they are catering more to the general consumer and less to developers. Thus, you really need to spend some time setting up your Mac to make it productive for power users. But for the moment it remains a very good compromise between Linux-like and Windows-like and thus forms the basis for our workstations.

Preliminaries

Here is how to purchase a computer at the lab. Before we begin, a few notes:

- In terms of the mechanics of purchasing:

    - use LBNL Ebuy (not Ebay) wherever possible - you need to be on the lab network (onsite via an ethernet cable) or be connected via the VPN
    - use Amazon, etc. to buy various components if not available via EBuy
- The laptop is government property; you are expected to return it to the group when you are done working at LBNL. Note that Mac computers make it very simple to transfer everything over to your next computer.
- You are free to take your laptop home, on trips, etc., unless you are an intern in which case other restrictions may apply from the internship program.
- The lab receives your computer and tags it before sending it over to you.
- You **must** back up your computer very regularly (at least once per week, ideally continuously). This is simple using the Time Machine app. Just plug your backup drive into your monitor so when you connect to your monitor, you also back up. If there are (for some reason) errors in backing up, fix that issue immediately. There are zero excuses for not doing this.

Selecting a computer, monitor, and accessories

Your computer workstation is one area where you should just get whatever you think will make you most productive and not care about cost. Seriously, just get what is best and do not worry about cost.

For the computer, you should select a Macbook Pro (any screen size) as mentioned above. You can use the Apple website to browse details.

Anubhav uses a 13" Macbook Pro. It is powerful enough to do serious work and light/small enough to use on a plane. A 15" Macbook Pro is also a good choice. If you would like to get anything other than a Macbook Pro, talk to Anubhav.

For the monitor, Anubhav uses a single Thunderbolt display but this is no longer available. One option available is the LG 27MU88-W (4K resolution) monitor which is on Ebuy. Note that one big screen is usually better ergonomically than dual monitors, and you can use the "Spaces" feature of Mac OS/X to quickly flip between virtual screens if needed (this is what Anubhav does).

For accessories, make sure to get:

- An extra charging cable
- A VGA adapter dongle
- An ethernet cable adapter dongle
- A Time Machine hard disk (for backup), I have currently use the *Western Digital 4GB Passport for Mac*
- A keyboard. I suggest Apple Wireless Keyboard since I like the feel of Mac keys and I also like a consistent feel between my laptop keyboard and my desk keyboard. If you prefer a larger or ergonomic keyboard, you can get that.
- A mouse/trackpad. I suggest Apple Magic Trackpad. Note that I've found that a mouse is better on Windows but a trackpad is better on Mac. The reason is because the Mac OS has really customized a lot of the interface for the trackpad (e.g., gestures). I also value consistency between my laptop and desk workstation. After awhile you get used to doing everything on

your trackpad even if you were previously very productive/accurate with a mouse on Windows.

- (optional) A presentation tool, *e.g.*, Logitech R800

Making the purchase

1) Provide all the details of your selections in an email and send to Anubhav. If all looks OK, he will give you a project and activity ID.
2) Go to eBuy, and for items available there, add them to your cart and submit the requisition with the project and activity ID, and SAS approver as Micah Liedeker.
3) For items not available on eBuy, contact esdradmin@lbl.gov (and cc Anubhav) to obtain a procurement form. Fill it out with item details (Vendor, website, price, etc.) and send it back to her.
4) If you select the overnight shipping option (ask Anubhav about this and the related extra costs) most parts, except the computer, will arrive within a week to 10 days. The computer needs to be tagged by the lab, so with overnight shipping, it should arrive within 2 weeks. Ideally, you will select your computer well before arriving at the lab and won't need overnight shipping.

Appendix C: Setting up a new Macbook

Upgrade your OS

If your computer is not using the latest OS, you should upgrade to the latest OS first.

Installing Python development environment

The best way to manage Python installations these days is a "conda env". This will allow you to manage different Python "environments", where each environment is a set of libraries that you have installed. For example, you can have one environment that uses Python 2.7 and has certain library versions installed, and another environment that uses Python 3.5 and has other libraries installed. Another advantage of conda environments is that you can apply the same procedure on NERSC and other computing centers that support conda.

How to do this:

- Follow the online instructions on installing a conda environment and see modifications below:
    - ***http://conda.pydata.org/docs/using/index.html***
    - (probably) prefer to install the "miniconda" version rather than anaconda
    - (probably) prefer to install "miniconda 3" rather than "miniconda 2". Both will work fine and allow you to do everything the other one does so don't stress too much about this decision.

- (probably) When creating environments, create at least one python 2 environment using the "python=2" parameter. It is up to you whether you want to work in Python 2.x or Python 3.x. For the moment, Anubhav prefers 2.x backward-compatible code for most base libraries (e.g. FireWorks), so if you use 3.x as your main environment, make sure your syntax doesn't depend on the newer features. See next bullet point for more.
- When creating environments, use a command like this (note that this also installs recommended libraries):

```
conda create --name py2 python=2 numpy
matplotlib pandas flask pymongo scipy
scikit-learn jupyter plotly
```

- If you want a reference guide to conda commands, try: ***http://conda.pydata.org/docs/using/cheatsheet.html***

Install high-throughput computation environment

Our group has a set of base codebases used for performing high-throughput calculations. Note that if your project does not involve high-throughput calculation, you may need only one or two of these libraries installed – contact Anubhav if you are unsure.

- Install the following packages using a combination of `git clone >>REPO_NAME<<` and `python setup.py develop`. Start with:

    - `git clone https://www.github.com/materialsproject/fireworks`
        - You might need to generate an ssh key for the git clone command to work:
            - `ssh-keygen -t rsa -b 4096`
            - no password is probably OK unless you are security conscious
            - add your SSH key to your Github profile
    - Then: `cd fireworks; python setup.py develop`
    - Repeat the process above but replace "fireworks" with:
        - pymatgen
        - pymatgen-db
        - custodian
        - atomate (note: this is on the hackingmaterials github site)
        - matminer (note: this is on the hackingmaterials github site)
- If you want, you can automatically source activate your environment in your .bash_profile file. This will automatically load your environment when you open a Terminal. Otherwise, you will start off in your default Mac Python and will likely cause you a lot of confusion

Configure Pycharm IDE

An IDE allows you to be a much more productive coder. It is like a text editor but contains many useful keyboard shortcuts, code-completion

tools, refactoring tools, and debugging/profiling tools to help you be more productive.

- Download Pycharm professional
- Get an activation license from Anubhav
- Configuring PyCharm - unfortunately no written instructions in this version of the Handbook.

Note that there are some advanced programmers that know their way around an IDE but still prefer an editor like vi or emacs with appropriate plugins. This is fine so long as you have first tried an IDE for a few months and really tried to make use of it. It is not fine if you are simply comfortable with other tools but never seriously tried using an IDE.

Other things to do

- Set up your Time Machine backup (make sure you have purchased or received an external hard disk).
    - ***https://support.apple.com/en-us/HT204412***
- You can also set up an online backup plan (e.g., Crashplan or Backblaze) to provide you with a second backup.
- Install MongoDB.
- Purchase Microsoft office from LBNL software distribution. Anubhav currently prefers Office 2011 due to stability and usability issues in the new subscription version.

Appendix D: Some notes on using a Mac from Anubhav

Basic setup

- Macbook Pro 13" laptop
- Thunderbolt Display (now discontinued)
- Apple keyboard
- Apple Trackpad - less precise than mouse, but can be very productive if you learn all the gestures (e.g., for web browsing back/forward, for mission control, for swiping between different Mac "Spaces")

I use an Apple keyboard and Trackpad so that typing/navigating is similar whether I am at my workstation or whether I am on my laptop.

Early on, I turned up my Trackpad speed all the way to the max. This means I can very quickly move the cursor all the way across the screen. It took a few days to get used to this very sensitive setting but now I don't even notice it (when other people use my trackpad, they usually freak out…)

There are many options I set to make OS/X more oriented for power users. For example, my Finder window shows directory paths at the bottom, I have sidebar shortcuts many important locations, I display hidden files, I have a shortcut to copy the path of the current Finder location to the clipboard, etc. There are many settings like these for

various built-in OS/X apps, but unfortunately I don't remember them all. Getting a good Finder setup is probably the most important.

Apps I use for programming

- I use the PyCharm IDE. Things I like about PyCharm include:
    - underlining errors
    - underlining code "lint", e.g., spacings that do not follow PEP
    - code highlighting / editor features (e.g., when you open a CSS file, lines of code that define a color automatically display a swatch preview of that color)
    - a nice and powerful search tool (regexes, find/replace in certain files, easily filter through results visually and categorize by what type of file they occur in)
    - autocomplete (ctrl+space)
    - autofix errors, i.e. red underline stuff (option+enter)
    - follow definitions of variables, methods, classes (Cmd+b)
    - quickly open classes (Cmd+o) and files (Cmd+shift+o) and variables (Cmd+option+o). Or simply tap shift twice to search across everything.
    - go back to previous/next file being edited like forward/back on a web browser (Cmd+[or Cmd+])
    - the "find usages" command
    - quick documentation lookup (F1)
    - code refactoring
    - structure view of code. I usually have "Project" view at left of window, code in middle of window, "Structure"

view at right of window, and "Todo", "Terminal", and "Python console" at bottom of window (along with search results).
    - debugger (sometimes)
    - easy IPython console to test code snippets
    - there are other commands that I use, but those are the ones I use most often
    - note that others use the git integration, which avoids needing to leave PyCharm to pull/push/etc., but I prefer Gitbox
- MongoHub (for visually exploring Mongo databases)
- Gitbox (I almost never use the Git command line; Gitbox is unique in that it is really easy to preview changes to the remote before pulling them in. It is also the most intuitive Git tool I know of)
- Patterns (for tricky regexes)
- Cocoa JSON Editor (for examining large JSON)
- Balsamiq Mockups - wireframes

Apps I use for Science

- CrystalMaker (and sometimes Vesta) - crystal structure visualization
- Mendeley - reference management
- MS Office Suite

Note that I use Mendeley not only for reference management but also for taking notes on articles. To take notes on articles, I first "star" the article and then use the "Notes" tab to take notes in plain text in the free

text box. Mendeley also allows you to directly take notes on the article PDF but I don't use that feature. Some of the things I like about this system:

- The notes are kept together with the articles, so I can quickly bring up the article if I am reviewing the notes
- I can easily see which articles I took notes on by selecting my "Favorite" (i.e., starred) articles in Mendeley, i.e. to browse the articles I have read and annotated before. I can also search/filter those annotated articles using keywords from my notes as well as the full text search via Mendeley, e.g., to see all the battery papers that I have taken notes on.
- The notes are very quickly readable as plain text (versus hunting for notes on the PDF itself) and I can export them easily via BibTeX export. This retains all the notes I took in the BibTeX in case I need to migrate to another system later.

This system isn't perfect but has worked well enough so far.

Apps I use for working more quickly

- Alfred - application launcher, quick file opening, quickly go to a web site. Note that if you don't use Alfred, the built-in Mac Spotlight now includes some of its features.
- Trickster - for easily calling up recent files, e.g. drag a recent file from Trickster into an email
- Default Folder X - the most useful feature of this is that it can add a sidebar to your save dialog that lets you access recent folders. This is 95% of the time where I want to save something.
- Fantastical - for quickly scheduling meetings or looking at my schedule

Apps I use to keep things organized

- Evernote
- 2Do - allows for complex todo lists, but also easy to use and intuitive. All my tasks are managed here.
- Screenshot Plus - Mac widget for quickly capturing screenshots (if like me you can't remember the keyboard shortcuts)

Misc Apps I use

- Bartender - allows you to clean up and reorganize your (top) menu bar; especially useful on a 13" screen
- ShiftIt - keyboard shortcuts for half-screen, full-screen, etc. like Windows has had since Win7
- Mousepose and IMovie - screencasts
- Tomato One - if I find it hard to be productive or am avoiding doing something, I revert to Pomodoro method with 40 minute sessions and 10 minute breaks
- Focus - for sometimes restricting internet browsing if I really can't focus (usually combined with Tomato One)
- Safari for web browsing (I find the experience to be very visually smooth and pleasing, e.g., when paired with Trackpad Gestures. For example, a two-finger pinch shows all tabs in a window.)
- Pocket - for saving web pages to read later, and then usually never getting around to it
- Time Machine - not only for backups, but also for sometimes recovering past versions of files that might have gotten accidentally changed / overwritten.
- CrashPlan - online backup (also consider BackBlaze)

- Inbox When Ready - a Chrome extension that helps control the flow of your email (requires checking your GMail via Chrome)
- Spotify - music
- Pixelmator - image editing

Appendix E: Our open source software philosophy

"If you want to go fast, go alone. If you want to go far, go together".
- Attributed to an African proverb

Although we develop both open and closed source pieces of code in our group, we try our best to release any software that is potentially useful to more than one person as open source. This ends up being almost all the software that we write except perhaps code written to conduct a specific scientific analysis.

Benefits of open-source software include:

- authors can include the code in their portfolio for future job applications
- you get recognition from the community of users of your code as well as personal pride
- more users means more bug reports - this sounds scary but is in fact very useful and important for your own research
- outside developers can contribute fixes and features, so your code gets better for free
- much less friction - easy to share code, fork it, etc. without needing to set up permissions or access. Easy to distribute and install the code, e.g. via PyPI

- many services like CircleCI and PyCharm offer their products for free when the codebase is open source. Not only does this save money, it more importantly saves a lot of time in coordinating purchasing requests that need to be renewed.
- your own programming will automatically improve because your code is open source and public. You will be more likely to write documentation and write clean code if you know it is for the world and not only for yourself. This will also encourage writing the code in a more general manner rather than specific to your application.
- you can write a paper about your code whenever ready. There is no separate process of "making the code open source" if it is already open source from day 1.
- it is the right thing to do for the betterment of the research community!

Clarifying common misconceptions about open source code:

- Writing open source code almost never exposes you to getting scooped or having some outsider leapfrog you in research. First of all, it is very rare that an outsider will use your code rather than make their own, especially if you do not advertise your code. Most of the time, you will have the opposite problem - i.e., to convince people to use and trust your code. Second, as the code author you are the expert in the code. Even when there is an outside user, it is rare that they are as proficient as you in the use of the code. Third, the majority of people are friendly and not as manipulative as you may think.
- Open source code doesn't need to be perfect, nor does it even need to be any good. Often people think that they will make a

code open source when it is "ready". This is not the right approach; code does not need to be "ready" to be open source.

- Publishing a code as open source doesn't mean that you need to support the code or vouch for its correctness. You are offering the code publicly without any guarantees whatsoever, and you don't have any additional obligations to anyone. However, if you actively want your code to be used by the community and extended, then be prepared to document and support your code, and to help users and resolve their problems. But this is a separate decision. It is perfectly OK to have an open-source code for which you provide no support so long as you don't try to advertise it for more than it is.

Appendix F: 10 ways to write better code

There are many, many books and articles on writing better Python code. Please use those if you want to really desire to become a good programmer. Here, I am just focusing on some of the most basic things that I think are particularly relevant to the types of scientific programmers we get in the HackingMaterials group.

Note: I used this site: ***http://markup.su/highlighter/*** to help write code blocks with coloring. For additional flair, you might also try using: ***http://instaco.de***

1. **Prefer data structures that don't require memorizing array indexes.**

Don't use a data structure (like a list/array) that requires one to remember that "index 8" is the species string and "index 1" is the coordination number.

Bad:

```
my_data = ["Fe2O3", 6, 5, 43, 4.1]
cell_volume = my_data[1] * my_data[2] * my_data[3]
is_insulator = my_data[4] > 3
```

Better:

```
my_data = {"formula": "Fe2O3", "a": 6, "b": 5, "c": 43, "band_gap": 4.1}
cell_volume = my_data["a"] * my_data["b"] * my_data["c"]
is_insulator = my_data["band_gap"] > 3
```

Notice how much easier it is to follow the logic of the code in the second example?

You can also use a pandas *DataFrame* object if you have lots of data and don't want to repeat the same column headers many times.

2. Document all classes and methods in a standard format

It is really important that all classes and methods are documented. Code is much more often read than written (a tenet of Guido van Rossum), so it needs to be readable and understandable. If you don't know what format to use, try the below:

http://bit.ly/2nAxlT0

You should also pay attention to the format already being used by a particular package.

3. Inside of classes/methods, write code that is readable without documentation whenever possible

This is usually achieved by writing descriptive variable names, function names, and good interfaces to functions. As a small example, why do this (requires documentation to tell user what my_d represents):

```
my_d = {"Mg": 3, "Ag": 8, "Li":4}  # dict of el. symbol to coord. number
all_element_symbols = my_d.keys()
all_coordination_numbers = my_d.values()
```

when you can do this (same clarity in first line, better clarity in last two lines, no documentation):

```
elsymbol_coordnum = {"Mg": 3, "Ag": 8, "Li":4}
all_element_symbols = elsymbol_coordnum.keys()
all_coordination_numbers = elsymbol_coordnum.values()
```

Of course, sometimes you will need to write documentation - but usually to explain why, rather than how. Here is the perfect article about that - it is short and sweet:

http://bit.ly/2pgFQXs

Read it!

4. Follow PEP formatting guidelines

Following proper code formatting helps clarify your code. There are a billion PEP rules and you don't have to follow all of them. But at least get the basic ones correct. Like:

- functions/methods are named like this: `my_very_first_method()`
- classes are named by CamelCase like this: `MyVeryFirstClass`
- python files are named like this: `my_very_first_file.py`
- python modules are named like this: `my_very_first_module`

If you use an IDE like PyCharm, it will detect, underline, and automatically fix most of the worst cases for you, so learn to use the feature. There are also tools like PyLint that you can use separately from IDEs (PyCharm basically has a nice wrapper around PyLint).

Also, don't use ugly code separator comments like "`##########################`" or ASCII art - stay clean and professional.

5. Use standard file formats

Use JSON or YAML most of the time if you need a file format, *e.g.*, for a settings file. XML is very heavyweight and quickly being outdated. Don't invent your own strange conventions (like CIF or any other custom file format).

6. Wrap exe code in 'if __name__ == "__main__":'

Python often runs your file even when you don't intend it to, e.g. when loading a module or importing some component of your file. It is important that you don't run code as a "side-effect" of this. Use the `if __name__ == "__main__"` wrapper to prevent this.

7. Be aware of Python gotchas, in particular mutable default arguments

Do you see anything wrong with this?

```
def append_to(element, to=[]):
    to.append(element)
    return to
```

If you don't see it, then you're going to get hit with some strange and difficult to pinpoint bugs downstream in your code.

This is a common Python gotcha (there is lots of discussion online about it)
http://bit.ly/2nijUdp
http://bit.ly/1wfFNKa

8. Write unit tests

Scientific researchers often don't write tests because (i) they don't write large, complex code with many moving parts or many different authors, (ii) they are overconfident about their ability to write correct code, (iii) they feel this will slow them down. Professionals write unit tests because they know that the longer and more complex a codebase becomes, and the more users it has, the more likely that something is going to go wrong down the line and the greater the dividends that are paid from writing unit tests. Unit tests allow code to be automatically tested for bugs every single time anyone makes a commit (continuous integration) and has demonstrated its value many times over in the large production codes that we use and develop - even (and perhaps especially) for ones required to do complex tasks on a deadline.

9. Throw exceptions rather than returning coded results

One of the most common beginner mistakes is to think that their code should never throw Exceptions or Errors. Perhaps this is because in the beginner's mindset, Exceptions are associated with bugs (e.g., they run a code with a bug and see an Exception, so Exceptions are bad). Another issue is that beginners never want their code to interrupt the operation of whomever is running it. So rather than throwing an Exception when their code is given bad inputs, they will return None, -1, or False, so that they don't interrupt whomever is calling their code.

This is bad. If the user gives bad or nonsensical input to a function, an exception needs to be raised and the program needs to stop immediately if the user is not explicitly catching the exception. For example, if you try to use the Python math library to compute the log of a negative number [`>> math.log(-1)`], it doesn't return None or some nonsense like False or -1. It throws an Error! As a user, the error is much more useful than any other course of option. Think of the alternatives for math.log(-1):

- If the function returned -1, you would have returned an incorrect result (extremely bad). For example if your function computed math.log(x) * 3, and you gave a negative x, your function would return -3 - which looks perfectly reasonable but is completely wrong! This is the worst possible thing you can do.
- If you return None or False to avoid inconveniencing the user, you have just made two mistakes. First, the user doesn't really know that their input was bad; perhaps it is simply the math.log() function has a bug leading to the strange output. The second and more important issue is that a user might want to

> run the math.log() function over an array of 1 million integers, and then do a lot of complex processing after that. If math.log() didn't immediately throw an error when encountering a negative number in the 1 million integer array, then the code would keep proceeding with nonsensical results and the user might finish 5 or 6 additional processing steps downstream before the code finally chokes and dies because there are strange "Nones" or "Falses" where there should have been numbers.There are even more dangerous situations that can occur, like a second library to compute standard deviations that ignores None values in the array. Then the user has unwittingly taken the standard deviation of only a subset of the data and never even knows that there was a problem in the pipeline. At that point, it becomes extremely tedious to trace back the source of the error.

Code should fail immediately when there is invalid input. In general, the further the "distance" from the actual place where the problem originated and the point of failure/exception in the program, the more difficult and maddening the debug task. You are doing users a favor by moving program failure right to the place of the problem.

10. Prefer python lists to numpy where practical

Numpy is great, but it is often overused (leading to worse code). Numpy is great for algorithmic work, for very complex slicing of multidimensional arrays, and for a host of other things, but it is *not as good for creating basic data structures*. Here are some advantages that Python lists have:

- Python lists have cleaner built-in functions and code. There are lot of tools for Python lists, like index slicing and iterations, or functions like sum() and all() that make them very powerful while still very clean. Numpy has even more useful functions and operations than that (e.g., a built-in mean()), and sometimes you might need numpy in order to leverage those features, but there is no need to transform to numpy arrays to (for example) take the sum of an array. Master regular Python lists first before reaching for numpy because you will have much cleaner code.
- Python lists can be easily appended and modified without lots of "filler" like figuring out how long the array needs to be in advance and populating with zeros before modifying values. This again leads to much cleaner code and is much easier to write and to read.
- Python lists can be easily serialized and deserialized, e.g. to JSON format where they are native.
- Despite the claim that numpy is fast, numpy lists, arrays, etc can actually take a lot of time to initialize - maybe 100X more than default Python. Of course, if you are then going to heavy processing on that matrix, like diagonalizing a large matrix or doing large matrix multiplications, numpy will absolutely improve your overall performance, perhaps to large degree. But for simply creating a data structure or taking the sum of a list, you will perform much worse with numpy while writing less readable code.
- Python lists are more universal; they don't require dependencies and they are readable by many more programmers.

Note that this doesn't mean to stop using numpy. Numpy can certainly do lots of things that regular Python cannot and it is an extremely powerful and useful library. But for routine file parsing (where being able to append easily is important), data representation (where serialization is important), overall code clarity (always important), and even speed for routine tasks (usually important) the native Python lists often have the advantage. So if Python lists are a chef's knife, and numpy is a swiss army knife, I am just saying that if you need to chop some onions stop using your swiss army knife. It does not make you clever to be able to do that. Learn to use the chef's knife until you actually need to open a can or do some other complex operation.

Appendix G: Giving effective presentations

"I am a successful lecturer in physics for popular audiences. The real entertainment gimmick is the excitement, drama and mystery of the subject matter. People love to learn something, they are 'entertained' enormously by being allowed to understand a little bit of something they never understood before. One must have faith in the subject and people's interest in it."
- Richard Feynman

"I would drop everything to hear him lecture on the municipal drainage system."
- David Mermin, about Feynman

Good presentations have a thoughtful purpose

The first step to giving an effective presentation is to truly understand their value. Here are just some of the ways in which presentations can be purposeful:

- to establish your expertise and to have your name recognized by the people in your field, especially independent to that of your supervisor
- to encourage people to collaborate with you
- to convince people to test your theory/prediction or to influence the research direction of others
- to convince someone or a committee to target you for a job offer or offer you funding for your idea
- to encourage people to cite your paper
- to encourage people to use or contribute to software that you've developed
- to receive useful feedback on preliminary ideas you may have
- to "test" the talk itself, i.e., gauge audience reaction and points of confusion (based on the after-talk questions) to improve subsequent presentations (like a stand-up comedian at a small venue)
- to simply help expand your audience's knowledge about a particular subject and "tell them something you've learned"
- to solidify your own thoughts about a topic!

Before you begin designing your presentation, you should be clear about one or more very well-defined goals you want to achieve by giving the presentation. For example, if you want to encourage people to use software you've developed, you'll need to include slides explaining its capabilities and benefits to the community and as well as how to obtain

and perhaps use the software. If you want people to test your theory, you should include slides suggesting how and why people might attempt this. If you want feedback on your ideas, you should further emphasize points of confusion / unresolved problems.

Note that you should not underestimate the value of giving presentations simply to establish your expertise and to promote your work. You may think that doing good research or writing a paper is enough. Unfortunately, this is usually not the case. Take the example of musicians: they cannot simply record an album and sit back expecting a devoted following of fans. They must instead earn their fans by going on tour and generating excitement about their music, perhaps starting out as a small and relatively unknown "opening act"; if they do a good job, the live act will encourage people to investigate the recordings. It is not much different for science; when you obtain a valuable result, you should go "on tour" and focus on disseminating it.

In summary, first decide on your goals for giving a presentation, then design your presentation around those goals.

Three good presentations

There is no single good presentation style. A good presenter doesn't have to be authoritative or have a low voice. You don't have to change your natural personality to give a good presentation. Here are some examples of people with different personalities nevertheless giving effective presentations.

- Donald Sadoway (15 mins): formal, authoritative, high salesmanship yet also unconventional/creative with well-rehearsed spontaneity (e.g., use of blackboard): ***http://bit.ly/1WuEkeK***
- Walter Alvarez (22 mins): approachable, casual / unpolished yet poetic, with several tangents - yet inspiring wonder in the subject: ***http://bit.ly/2ov568E***
- Mick Mountz (12 mins): not necessarily a "natural speaker", but makes a boring subject (packing boxes) fascinating through a great presentation structure and slides: ***http://bit.ly/2oSwO1y***

None of the talks are perfect, and thinking about why will help your own presentation skills. However, the above talks are able to get the audience interested in the problem and invites them to briefly join them in their field of study. This is in contrast to talks that try to oversimplify concepts or try to sugar-coat them with fancy graphics - i.e., present problems solutions in a way that is different than the way they themselves think about it. This is a mistake that many other TED-style talks or cable TV documentaries about science make. These talks also include small tangents that could easily be the subject of other talks. You can learn a lot from seeking out and taking notes on good presentations.

Presentation checklist

Here is a checklist you can use to improve and verify various aspects of your presentation.

Easy things to do:

- ❑ **Number your slides.** Numbered slides make it easy to refer to specific slides during the Q&A or feedback period.
- ❑ **Confirm all font sizes are large enough so that even people in the back of the room can read them.** One good way to do this is to make the fonts way too big, then reduce the size until manageable (rather than starting too small and increasing from there, which in 90% cases leads to fonts that are still too small).
- ❑ **Write slide headings as snippets that contain useful information.** A bad heading would be "*Effect of +U parameter*". A good heading would be "*Band gap and VBM d-character increase with +U parameter*". There is a style of slide called "evidence-assertion" that is generally very effective and should be used often.

Intermediate things to do:

- ❑ **Minimize the use of written text.** Research demonstrates that your audience cannot read text on your slide and process what you are saying at the same time. Every second they are reading, they are not listening to you. In contrast, audiences have no trouble simultaneously listening and processing visual information (diagrams, images, etc.). Design your slides to account for this quirk.
- ❑ **Convey information through multiple "channels".** Ensure that critical information is not only contained in your speech/delivery but also through a visual channel (images or short text phrases / conclusions). People may not be able to hear you or might be distracted by their own thoughts for many moments in your presentation. Or, they might not understand a visual diagram and be helped by reiterating the point a different

way through your dialogue. Having multiple channels maximizes the chance that they will receive the signal of your talk even when there is external "noise". More advanced presenters will use body language or position as another "channel" through which to convey information.

- ❑ **Rehearse your talk for "flow", "momentum", and "energy" and cut slides that disrupt flow.** Rehearse your talk, paying attention to the slides/sections in the talk where you are (i) struggling to explain a slide, (ii) where your energy / enthusiasm level drops, or (iii) the momentum of the talk seems to be slowing down. One symptom of such struggling is talking quickly in order to explain everything on the slide. 90% of the time, I find that *removing* such slides from the talk (i.e., moving it to an Appendix/supporting slides) is the best course of action - even if I initially think that slide is important. Rehearsing the section again usually reveals you can maintain the energy and flow of your talk much better without the obstruction of having to explain that slide, and you can explain away the missing concept in a sentence or two while retaining the momentum of the previous slides. If it turns out the slide was in fact critical, then perhaps re-design the difficult slide as multiple slides to more gradually set up the concept.
- ❑ **Memorize the order of your slides; use "Slide sorter view" to help.** During every point in delivering your presentation, you should be able to picture what the next slide in the presentation is. If you can do this, you are more likely to speak in a way that naturally connects between slides rather than abruptly stops/starts between slides. Some presenters use "presenter view" in Powerpoint during their talk to help with this, but I

would say that depending on this feature is less likely to lead to smooth explanations than memorization. To mentally remember the order of slides, I stare at the slide deck in "Slide Sorter" view. The "Slide Sorter" view can usually show me most or all of the presentation at once since each slide is a small thumbnail, and I can easily see the visual overview of essentially the entire presentation. Thus, I can remember the visual arrangement of slides by studying the Slide Sorter view and can roughly flip through the presentation in my head.

Advanced things to do:

- ❑ **Video record yourself rehearsing the talk and watch yourself.** Although you may find this uncomfortable or strange, you will learn much about your presentation style and areas to work on.

Appendix H: Hands-on exercises for machine learning in materials

There are many excellent tutorials for learning machine learning in general with Python, and some of them were mentioned earlier in this handbook. Those might be a good place to start. However, if you want to dive into machine learning for materials science or prefer a more hands-on approach, you might try following some of the steps below.

Step 1 - git and Citrine tutorial

Skills gained: git, Github, Python, MP Rest API, scikit-learn

Use an IDE (e.g. PyCharm) and invest in getting it running and set up on your system. Force yourself to learn the IDE and its features over time.

- Create a new repository in your Github account (public is fine).
- Complete Parts 1 and 2 of Citrine Informatics's blog on Machine Learning for the Materials scientist.
- Push the code to your repository. Make sure to commit to git throughout at short intervals, i.e. at the completion of every step, i.e. at least 6 commits for the tutorial (more is fine).

Step 2 - pandas and data exploration

Skills gained: pandas, plotting, data exploration

Note that the steps below should be ***easy***, *i.e.*, built-in functions in pandas for the most part that can be done in 1-2 lines of code. If you find yourself trying to write long and complicated code, you are probably not doing it correctly.

- When you are reading the data from the CSV file, read it into a pandas dataframe object.
- Use the data stored in the pandas dataframe to get summary statistics on the band gap data. min, max, mean, quartiles, etc. (look for the built in pandas function to get summary statistics)
- Make a histogram plot of the band gap data distribution.

- Add to your data frame - make a new column that is equal to the square of the band gap.
- Make a quick scatter plot of gap vs gap^2.
- Use the scatter_matrix() function to see the relationships between all quantities

Step 3 - further explorations with pandas

Skills gained: putting data into a pandas dataframe

- Rewrite the Citrine tutorial in Step 1 above (i.e. scikit learn fitting, adding features to the data, etc.) using pandas as the data structure. e.g., when you add features, it should be a new column in the data frame. What this means is that the "materials, bandgaps, and naive_features" lists should be *deleted* and unnecessary, and all 3 of these variables replaced by the pandas data frame.

Step 4 - playing with different models

Skills gained: bringing all the skills together to do a new analysis

- Get the formation energy in addition to the band gap for each compound using MPRester and put it in the data frame. So now you have columns for gap and formation energy (in addition to the one for material).
- Test if also including the formation energy in your predictor set improves your model of the band gap.
- Also repeat the above with the density of the material.

- Compare the baseline errors of the different models that use the naive and physical feature sets when using just the bandgaps, formation energies, and densities, and in different combinations with each other.
- Make a quick scatter plot to visualize the correlations, if any, between the properties and compare with the baseline errors above.

Step 5 - the interactive Jupyter notebook

Skills gained: Jupyter notebook

- Create a Jupyter notebook for your code. The final product should be a nice document with all plots in-line and that clearly show all steps.

Step 6 - matminer

Skills gained: the matminer package that we developed.

- Install matminer on your system, if not already installed
- Work through the example Jupyter notebooks for matminer

Finally, give yourself a pat on the back - and think of how you can now apply your newfound skills!

Appendix I: Managing the group web site

It is currently a two-step process to update the group web site:

- update the code in the hackingmaterials.github.io repo
- push the code to our GoDaddy hosting account, e.g., using FTP to GoDaddy

In the future, it would be better to bypass the GoDaddy hosting step by hosting the website entirely on Github using Github IO pages: ***https://pages.github.com/***, and then pointing it to the group domain (hackingmaterials.lbl.gov) using the guide: ***http://bit.ly/2mBhxOl***
To update the hosting itself, and handle things like SSL certificates required by LBNL, Anubhav should have some details on the current hosting in an email with subject "two things regarding group web site" with a summary by Saurabh.

Appendix J: Group library

We have several technical books in the group that you can borrow (just contact Anubhav). The only condition is that you should only borrow books when you intend to read them. It is all too easy to take a book with the *intention* to read it and let it simply take up space on your desk for a year. Thus, the maximum borrowing period is ***one month***, which is certainly enough time to go through a book once you've actually decided to read it. After that, you'll need to wait a week before borrowing again, and someone else is free to take the book in the meantime. You can also

instead borrow a book for two days for any reason - e.g., to skim it over or to use it as a reference for understanding some concept - and as long as you return it afterwards it doesn't affect your ability to "check it out" later. The main thing to avoid is just having a book sit on your desk for 3 months.

Solid state physics / thermodynamics

- ***Principles of Electronic Materials and Devices*** by Kasap
    - A good beginner text for solid state physics - often can help pave the way to understand more difficult texts
- ***Fundamentals of Semiconductors*** by Yu and Cardona
    - A good intermediate introduction and reference to solid state physics concepts
- ***Density functional theory: a practical introduction*** by Steckel and Sholl (2 copies available)
    - My personal favorite introduction to DFT
- ***Fundamentals of Carrier Transport*** by Lundstrom
- ***Quantum Transport: Atom to Transistor*** by Datta
- ***Lessons from nanoelectronics: a new perspective on transport*** by Datta
- ***Introductory statistical mechanics*** by Bowley and Sanchez
- ***Molecular Driving Forces*** by Dill and Bromberg
    - A great introduction to statistical mechanics, with plenty of conceptual explanations

Materials science / chemistry

- ***Physical Ceramics*** by Chiang, Birnie, Kingery

    - A nice overall introduction to (non-metallurgy) materials science
- ***Organic Solar Cells: Device Physics, Processing, Degradation, and Prevention*** by Kumar
- ***Chemical bonding in solids*** by Burdett
- ***Materials and the Environment*** by Ashby
- ***Kinetics of Materials*** by Baluffi, Allen, Carter
- ***Organic Chemistry I for Dummies*** by Winter
- ***Organic Chemistry II for Dummies*** by Moore & Langley

Programming / Data Science / Statistics

- ***Effective Python*** by Slatkin
    - A good way to bring your Python skills to the next level
- ***Data Science from Scratch*** by Grus
    - A good but short introduction to a variety of data mining techniques with Python; emphasis on "from scratch" is sometimes helpful, sometimes painful
- ***Learning pandas*** by Heydt
- ***Statistics in a Nutshell*** by Boslaugh and Watters
- ***Introduction to the Practice of Statistics*** by Moore and McCabe
- ***Effective Java*** by Bloch

Writing / presenting / professional skills

- ***Slide:ology*** by Duarte
- ***Scientific Writing and Communication*** by Hofmann
- ***The MIT Guide to Science and Engineering Communication*** by Paradis and Zimmerman

- ***The Craft of Scientific Presentation*** by Alley
    - There are some nice stories and good amount of background research in here, but the actual practical application is hard to access unless you read every word. The best use of your time is just to read the figures and captions for Chapter 4, (pages 105-212) - these figures show some examples of bad slides vs. better slides.

Thank you!

Thank you for contributing to this handbook!

- Saurabh Bajaj
- Alireza Faghaninia
- Joey Montoya
- John Dagdalen
- Nils Zimmerman
- Ben Ellis
- Maksim Rakitin