Getting Help

We *all* need help from time to time—and that's what we're here to provide—but the best way to 'get help' will always be taking steps to 'help yourself' first!

Helping Yourself

Here are four things that you can do to 'help yourself' *first*:

1. Use Google–this is one course where saying "I googled it…" will be taken as a good sign! Probably the biggest difference between a good programmer and a new programmer is that the good one knows which terms to type into Google to get the answer that they need right away.
2. Use Stack Overflow–as you become a better programmer you'll start to understand how to frame your question in ways that produce the answer you need in the first couple of search results, but whether you're a beginner or an expert Stack Overflow is your friend. *True story*: I have sometimes found answers that *I* provided (but didn't remember giving) when trying to solve a problem.
3. Use the work of others–beyond the readings that we've assigned to support the module's learning outcomes there is a world of knowledge out there on which you can build! Use Google Scholar, Medium, and dedicated tutorial-type sites like Towards Data Science and Programming Historian to see if you can find others who have had similar challenges.
4. Use the dedicated Slack channel–this provides a much richer experience than the Moodle Forum and should be your primary means of requesting help outside of scheduled teaching hours because you can get answers from other staff, other students, and the wider CASA community.

Yes, this is a lot of things to do when you want to know the answer to what feels like a simple question, but it's an investment: if we just 'give' you the answer then chances are you'll forget it as soon as your code starts running again, but if you've had to invest your time and energy is sorting through a whole range of answers (some useful, some not) then not only have you found it *for yourself* in a way that you'll not soon forget, but you've also learned something about how *frame questions* and *identify useful answers* when we're not around to help you out. That, frankly, is a much more valuable skill!

Creating Opportunities

Learning to code is like learning a language: you need to practice! Set yourself little problems or tasks and see if you can apply what you've learned in class to a problem in a *different* class, or to a friend's problem, or just something you're curious about! In the same way that practicing your Chinese or French with native speakers will help you to learn those languages, so will practicing your Python.

How We Can Help

If you've gone through steps 1–4 and don't feel any closer to a solution then it's time to get our input! So the final steps:

5. Make use of 'Office Hours'–if you are struggling, then tell us! We can't help you if we don't know that you're lost! That doesn't mean that we can simply 'give' you the answers to challenging questions, but we will do everything that we can to support you in finding and understanding the answers.
6. Email us–Slack will usually be faster, but for personal questions or ones you're just not comfortable asking in public, then email away!
7. Sign up for online classes–realistically, you will have a lot on your plate this year, but if you want or need more practice with Python then there is a wealth of options out there for 'further study' and 'further practice'. Perhaps you'll find a resource that speaks to you in a way that our module doesn't!

How to Ask for Help

However tired you are, don't send a stream of consciousness late-night email saying little more than "Hey, I'm stuck on this problem can you solve it?" Go to bed. Sleep on it. And if you're still stuck in the morning it's time for the email.

What does a *useful* email look like? You might want to follow this overview of how to get a busy person to respond to your email:

- Keep it short.
- Format it for readability and clarity.
- Make it clear what you want me to do.
- Be reasonable with your request.
- Show me why I should take the time to help you.

That last point isn't quite as rude as it sounds! If you've gone through steps 1–4 above, then it's actually easy to explain what you've done, what you've found, why you think things aren't working, and whether you have any ideas for solving your issue! If you've

done all this then your question will *never* be a result of laziness, so that suggests there's something for *us* to learn about how we teach!

In academia there ar ea few more things I'd add:

- What module are you emailing me about?
- What is your student id? (Especially if it's about Extenuating Circumstances or an Assessment)
- Is the question about a specific practical, lecture, or technical problem?

And here is some additional insight into how to email your professor (without being annoying AF):

- – *Salutation*: should I use "Dear", "Hello" or "Hi"?

- *Honourific*: should I use "Mr", "Ms", "Dr" or "Professor"? *Hint:* don't *ever* use Mr/Mrs/Ms.

- *Name*: for god's sake, please try not to get this wrong.

- *Exceptions*: always look at how your Prof. or TA responds to *you* for cues about how to respond to *them*.

- *Be nice*: treat your TA and Prof like human beings please!

- *Remind me who you are*: anything that allows me to place you and your question in context.

- *The reason*: tell me *as precisely as possible* why you are emailing me and what you want/hope to achieve by doing so.

- *Do the legwork* (this is the 'show me why I should take the time to help' thing above): if the answer is in the Syllabus or the recorded content then I may not answer your question. Or I may just write "It's in the syllabus" and leave it at that. Show me that you've *tried* to answer your question yourself and give me a sense of what you've already tried. For instance, if your problem is technical then "I couldn't install the software and it didn't work" tells me *nothing* about your actual problem (see also information on asking a good tech question and the links in Lecture 1.1).

- *Wrap-up and Sign-off*: is there a deadline (e.g. for a recommendation) or some other issue that I need to factor into my plans? Some recognition of thanks never goes amiss.

The Follow-Up!

If we don't reply to you ***then send a reminder***! The trick is to send the reminder at the right time: if you are about to fail an assessment or are three weeks into the course and still can't run the programming environment then this is urgent and you can send a reminder much sooner than if you're wondering 'whether X would be a good topic for the final assessment'. As the Medium blogger puts it: "If it can wait a week, let it wait a week" (before following up).

Why This Matters

We all have different 'registers' for speaking with other people: family (siblings vs. grandparents), friends (close friends vs. acquaintances), and so on. A professionally-written email is a vital work skill since most of the people that you will end up *working for* will use email over all the other channels available now. Sending an email like this will not get you the help you need:

> Hey Bossman,
>
> Are U awake at 2am like me? LOL. Having problems and not going to finish the report in time for meeting tomorrow. Can you give me an extension?
>
> Later.

If you're worried that you might not be hitting the right 'tone' then it's ok to say so and to ask if you write it better/differently. This shows reflection and thought, which is all that we really want from our students!

What Do *You* Want?

Finally, we want you to think about *how to learn* what you need to learn. This is called meta-cognition—thinking about thinking—and it's probably one of the most important study skills of all. So here you need to think about what *you* want from an assessment: do you just want to pass, or do you want to hit it out of the park?

Some questions to ask yourself:

1. For an exam: What kinds of questions might it include?
2. For an essay/written piece: What kinds of topics have we covered in class?
3. For *either*: identify key terms, define those terms, and question the question (what might be the assumptions behind it?).
4. Which of resources would help me to study/prepare?

5. How should I make use of lecture notes, practice exam questions, textbook and other readings, instructor office hours, peer discussions, and tutoring. Write down why each resource would be useful and how you could use it to map out a study/writing plan.
6. How does my essay work *as a story*? I don't necessarily mean a mystery or adventure novel, but a good essay has: a plot (what is sometimes called a narrative arc), characters (the key ideas from the literature, the data, the methods, the problems…), development (which character needs to be introduced first?), and a conclusion (what happened?). Have a look at the Tim Squirrel Guide where it talk about essay stucture.
7. Looking at this literature/piece of research, is this the way that I would have done it? If I would have done it differently do I think that would have been more, or less, effective? Why or why not?
8. Looking at my peers, is there someone who is doing really well who I could talk to about *how* they study? How they organise their time? How the make use of the resources (literature, etc.)?

Remember

When you are first learning to code there is *no such thing* as a stupid question. From time to time we all have lazy questions, which is what happens when we are frustrated and just want to know 'the answer' without putting in the work to clarify the problem. However, if any time you find yourself stuck on a particular problem there is a 100% chance that someone else in the class is having the same problem as well but hasn't quite worked up the courage to ask. So please: **ask**.