How to Ask a Technical Question

in a way that's likely to efficiently lead to a helpful answer

Bootcamp 2018

Instructor:
Dawn Koffman, Statistical Programmer
Office of Population Research (OPR)
Princeton University

Various Types of Technical Questions

- How do I do X?
    - I have no idea where to start
    - I have some idea of what tool(s) to use ... but don't know how to use them

- I'm trying to do X, but it isn't working, why?

Various Modes of Question/Response Communication

- In-person
- Phone
- Email to an individual
- Email to a group
- Post a question to a public forum

Goal of learning how to ask better technical questions

- **Quickly receive helpful responses**
- Nice side effect ...

  your ability to answer your own questions will improve

When asking for help, try to be an observant, active partner in developing a solution ...

you will learn from being fully-engaged in the “solution-finding” process.

Make it clear that even a pointer to an answer would be helpful.

Prepare your question. Think it through.

Follow the golden rule ...
imagine you are on the <u>receiving</u> end of the question,
and ask the question in a way that you would want to hear
(or read) it and respond to it

First, explain the overall goal of what you are trying to do

- it's possible that you are not taking the best path to reach that goal

Law of the instrument

natural human tendency to be overly-dependent on existing skill-set

IF YOU ONLY HAVE A HAMMER
YOU TEND TO SEE EVERY PROBLEM AS A NAIL
ABRAHAM MASLOW

People often prefer to use tools they're familiar with, rather than look for the best way to do something

Once you’ve confirmed that you are on the right path, then ...

1. describe the specific steps you are taking

2. carefully describe what isn't working, the symptoms of your problem
   - if relevant, provide screen shots or screen recordings

3. if you are seeing error messages, warnings or useful debugging info of any sort,
   say <u>exactly</u> what they are

4. describe your hardware and software computing environment:
   machine, OS, language, database type, applications, IDEs, commands
   and packages, API’s; include versions and release levels

5. describe any <u>recent changes</u> to your computing environment

Once you've confirmed that you are on the right path, then ...

6. explain the research you already did to try to answer your own question
   - keep track of what you find and learn
   - provide links to relevant external resources

     why?

7. explain the diagnostic steps you already took to try and pin down the problem

   why?

Once you’ve confirmed that you are on the right path, then ...

8. if at all possible, provide a way to reproduce the problem

   this does not mean attaching a lot of code or data to an email question,
   or asking someone face-to-face to go through a lot of code

   if you have a complicated piece of code that is not working,
   try to trim it to make it as small as possible

   effective way to be precise about a code issue:
   provide a minimal, complete, verifiable example

   useful because:
   - simplifying will more likely lead to a quicker and more useful answer
   - in the process of refining your issue, you may be able to see and solve the problem yourself

useful code example is:

- as small as can be ...
    - two strategies: start from scratch, or, remove code as long as problem remains
    - with either strategy: try to remove all code that is not related to the problem
- standalone, if possible ... for example, include smallest data file(s) necessary, or use built-in, example data if available ... particular input data is sometimes important
- something that (compiles and) runs ... but hopefully will be so easy to understand that running the code is not necessary
- tested and verifiable: make sure the code you provide actually reproduces the problem
- reproducible ... if example code includes a random process (could be for something as simple as generating a random sample of data), set the seed for the random number generator. Random numbers returned by a generator are a sequence of numbers that can be reproduced. Setting a seed informs the random number generator where to start.

providing a minimal, complete, reproducible example allows for an efficient conversation

- doing this has the side effect of making you think more clearly about your code

- doing this may be a lot of work, but the question-answer process is a partnership

- some research* has shown that providing a code snippet is correlated with a question being "successful" (on Stack Overflow)

Logistic Regression

	Coefficient estimate	p-value
Presence of Code Snippet ('No' as default)	0.71	<.0001

* Calefato F., F Lanubile, N Novielli, How to ask for technical Help? Evidence-based guidelines for writing questions on Stack Overflow. *Information and Software Technology,* 94 (2018) 186-207.

Be precise

explain exactly what you expected the code to do, and exactly what it is doing instead ...

"do you know why isn't this working?" is usually not a good question

for a graph, explain (or show) how you would like the graph to look

describe the raw symptoms of your problem in chronological order

no need to be self-deprecating ....

"I know this must be a very simple issue, and I'm not that good a programmer, but ..."

- this isn't particularly helpful
- much better to try to be as precise as possible about your question

for email questions ... please don't use all caps

After you've asked your question

If you don't understand the response to your question, don't immediately come back with another question. Do some research and experimentation to try to understand the response.

For example, if you are given a suggestion to use a linux stream editor called sed, that allows you to make edits to a text file regardless of file size, use man pages, FAQs, online examples, experimentation, etc, to learn how sed may help you with your particular task.

If you do understand the response, and it leads you to a solution, either directly or indirectly, let whoever helped you know this. Everyone learns this way.

Before you ask a question: Be resourceful and do your research

- if you are seeing a particular error message, Google the error message, *exactly as it appears*

- search the Web for information related to the topic; may need to do this multiple times if new to issue-specific computing terminology

- search relevant on-line help forums, such as Stack Overflow

- read the manual/official documentation

- read FAQ's related to the issue, blogs, tutorials, slide presentations, etc.

- if you are not getting the desired result/output, use inspection and <u>experimentation</u>; double-check your input data

- if you are using open-source tools, look at the source code

Before you ask a question: Be resourceful and do your research

This all takes time.

Don’t expect to solve a complicated issue after a few minutes of searching.
But learning how to search for solutions helps you become more self-sufficient.

If you don't find a solution, you will be <u>closer</u> to answering your question than you were when you started and you’ll be able to ask a <u>more informed question</u>.

But know when it’s time to stop and try to get help ...

If you’re not making progress, feel like
you’re going in circles, <u>it’s time to ask a question</u>.