Tutoring in computer labs

SoM tutor induction

Charlotte Desvages

January 11, 2023

School of Mathematics

Overview

1. Computer labs in the School
2. Preparing a computer workshop
3. Helping students during labs
4. Putting it into practice
5. Resources

Computer labs in the School

Computer labs in SoM

- Most are **programming labs**, typically Python or R (with some exceptions).
- Main opportunity for students to **practice** and get help from classmates/tutors.
- Sometimes in computer lab room, sometimes in "traditional" workshop room (with students working on laptops).
- Guidance for tutoring workshops also applies.

Computer labs in SoM

- Most are **programming labs**, typically Python or R (with some exceptions).
- Main opportunity for students to **practice** and get help from classmates/tutors.
- Sometimes in computer lab room, sometimes in "traditional" workshop room (with students working on laptops).
- Guidance for tutoring workshops also applies.

Computer labs in SoM

- Most are **programming labs**, typically Python or R (with some exceptions).
- Main opportunity for students to **practice** and get help from classmates/tutors.
- Sometimes in computer lab room, sometimes in "traditional" workshop room (with students working on laptops).
- Guidance for tutoring workshops also applies.

Computer labs in SoM

- Most are **programming labs**, typically Python or R (with some exceptions).
- Main opportunity for students to **practice** and get help from classmates/tutors.
- Sometimes in computer lab room, sometimes in "traditional" workshop room (with students working on laptops).
- Guidance for tutoring workshops also applies.

Questions to ask your Course Organiser

Different courses do computer labs differently, and for different purposes.

- **Learning outcomes** related to computing?
  - general programming skills
  - mathematical/statistical computing
  - proficiency with a particular piece of software
  - etc.
- **Software/platforms** students are expected to use in the course?
  - Install/set up your own machine with the same workflow
- Are students expected to **collaborate** during the workshops?
  - If so, how? (informally, in groups of 2–3, pair programming, ...)
  - If tasks are assessed, what is acceptable collaboration?
- **Marking and feedback:** what, how, and when?

Questions to ask your Course Organiser

Different courses do computer labs differently, and for different purposes.

- **Learning outcomes** related to computing?
  - general programming skills
  - mathematical/statistical computing
  - proficiency with a particular piece of software
  - etc. . .
- **Software/platforms** students are expected to use in the course?
  - Install/set up your own machine with the same workflow.
- Are students expected to **collaborate** during the workshops?
  - If so, how? (informally, in groups of 3+, pair programming. . .)
  - If tasks are assessed, what is acceptable collaboration?
- **Marking and feedback:** what, how, and when?

Questions to ask your Course Organiser

Different courses do computer labs differently, and for different purposes.

- **Learning outcomes** related to computing?
  - general programming skills
  - mathematical/statistical computing
  - proficiency with a particular piece of software
  - etc. . .
- **Software/platforms** students are expected to use in the course?
  - Install/set up your own machine with the same workflow.
- Are students expected to **collaborate** during the workshops?
  - If so, how? (informally, in groups of 3+, pair programming. . .)
  - If tasks are assessed, what is acceptable collaboration?
- **Marking and feedback:** what, how, and when?

Questions to ask your Course Organiser

Different courses do computer labs differently, and for different purposes.

- **Learning outcomes** related to computing?
  - general programming skills
  - mathematical/statistical computing
  - proficiency with a particular piece of software
  - etc. . .
- **Software/platforms** students are expected to use in the course?
  - Install/set up your own machine with the same workflow.
- Are students expected to **collaborate** during the workshops?
  - If so, how? (informally, in groups of 3+, pair programming. . .)
  - If tasks are assessed, what is acceptable collaboration?
- **Marking and feedback:** what, how, and when?

Questions to ask your Course Organiser

Different courses do computer labs differently, and for different purposes.

- **Learning outcomes** related to computing?
  - general programming skills
  - mathematical/statistical computing
  - proficiency with a particular piece of software
  - etc. . .
- **Software/platforms** students are expected to use in the course?
  - Install/set up your own machine with the same workflow.
- Are students expected to **collaborate** during the workshops?
  - If so, how? (informally, in groups of 3+, pair programming. . .)
  - If tasks are assessed, what is acceptable collaboration?
- **Marking and feedback:** what, how, and when?

Questions to ask your Course Organiser

Different courses do computer labs differently, and for different purposes.

- **Learning outcomes** related to computing?
  - general programming skills
  - mathematical/statistical computing
  - proficiency with a particular piece of software
  - etc. . .
- **Software/platforms** students are expected to use in the course?
  - Install/set up your own machine with the same workflow.
- Are students expected to **collaborate** during the workshops?
  - If so, how? (informally, in groups of 3+, pair programming. . .)
  - If tasks are assessed, what is acceptable collaboration?
- **Marking and feedback:** what, how, and when?

Questions to ask your Course Organiser

Different courses do computer labs differently, and for different purposes.

- **Learning outcomes** related to computing?
  - general programming skills
  - mathematical/statistical computing
  - proficiency with a particular piece of software
  - etc. . .
- **Software/platforms** students are expected to use in the course?
  - Install/set up your own machine with the same workflow.
- Are students expected to **collaborate** during the workshops?
  - If so, how? (informally, in groups of 3+, pair programming. . .)
  - If tasks are assessed, what is acceptable collaboration?
- **Marking and feedback:** what, how, and when?

Questions to ask your Course Organiser

Different courses do computer labs differently, and for different purposes.

- **Learning outcomes** related to computing?
  - general programming skills
  - mathematical/statistical computing
  - proficiency with a particular piece of software
  - etc. . .
- **Software/platforms** students are expected to use in the course?
  - Install/set up your own machine with the same workflow.
- Are students expected to **collaborate** during the workshops?
  - If so, how? (informally, in groups of 3+, pair programming. . .)
  - If tasks are assessed, what is acceptable collaboration?
- **Marking and feedback:** what, how, and when?

Questions to ask your Course Organiser

Different courses do computer labs differently, and for different purposes.

- **Learning outcomes** related to computing?
  - general programming skills
  - mathematical/statistical computing
  - proficiency with a particular piece of software
  - etc. . .
- **Software/platforms** students are expected to use in the course?
  - Install/set up your own machine with the same workflow.
- Are students expected to **collaborate** during the workshops?
  - If so, how? (informally, in groups of 3+, pair programming. . .)
  - If tasks are assessed, what is acceptable collaboration?
- **Marking and feedback:** what, how, and when?

Questions to ask your Course Organiser

Different courses do computer labs differently, and for different purposes.

- **Learning outcomes** related to computing?
  - general programming skills
  - mathematical/statistical computing
  - proficiency with a particular piece of software
  - etc. . .
- **Software/platforms** students are expected to use in the course?
  - Install/set up your own machine with the same workflow.
- Are students expected to **collaborate** during the workshops?
  - If so, how? (informally, in groups of 3+, pair programming. . .)
  - If tasks are assessed, what is acceptable collaboration?
- **Marking and feedback:** what, how, and when?

Questions to ask your Course Organiser

Different courses do computer labs differently, and for different purposes.

- **Learning outcomes** related to computing?
  - general programming skills
  - mathematical/statistical computing
  - proficiency with a particular piece of software
  - etc. . .
- **Software/platforms** students are expected to use in the course?
  - Install/set up your own machine with the same workflow.
- Are students expected to **collaborate** during the workshops?
  - If so, how? (informally, in groups of 3+, pair programming. . .)
  - If tasks are assessed, what is acceptable collaboration?
- **Marking and feedback:** what, how, and when?

Questions to ask your Course Organiser

Different courses do computer labs differently, and for different purposes.

- **Learning outcomes** related to computing?
  - general programming skills
  - mathematical/statistical computing
  - proficiency with a particular piece of software
  - etc. . .
- **Software/platforms** students are expected to use in the course?
  - Install/set up your own machine with the same workflow.
- Are students expected to **collaborate** during the workshops?
  - If so, how? (informally, in groups of 3+, pair programming. . .)
  - If tasks are assessed, what is acceptable collaboration?
- **Marking and feedback:** what, how, and when?

Preparing a computer workshop

Preparation

- CO will provide task and solutions in advance.
- **Try** the task yourself **without looking at solutions**.
  - Anticipate different ways that students might think about the task, and where they might get stuck.
  - Make notes of any useful bits of lecture notes, software documentation, or previous weeks' exercises to refer to.
- Ideally, keep the model solution only to check that results are correct. Students come up with lots of creative ways to solve a task – meet them where they are!

Preparation

- CO will provide task and solutions in advance.
- **Try** the task yourself **without looking at solutions**.
  - Anticipate different ways that students might think about the task, and where they might get stuck.
  - Make notes of any useful bits of lecture notes, software documentation, or previous weeks' exercises to refer to.
- Ideally, keep the model solution only to check that results are correct. Students come up with lots of creative ways to solve a task – meet them where they are!

Preparation

- CO will provide task and solutions in advance.
- **Try** the task yourself **without looking at solutions**.
  - Anticipate different ways that students might think about the task, and where they might get stuck.
  - Make notes of any useful bits of lecture notes, software documentation, or previous weeks' exercises to refer to.
- Ideally, keep the model solution only to check that results are correct. Students come up with lots of creative ways to solve a task – meet them where they are!

Preparation

- CO will provide task and solutions in advance.
- **Try** the task yourself **without looking at solutions**.
  - Anticipate different ways that students might think about the task, and where they might get stuck.
  - Make notes of any useful bits of lecture notes, software documentation, or previous weeks' exercises to refer to.
- Ideally, keep the model solution only to check that results are correct. Students come up with lots of creative ways to solve a task – meet them where they are!

Preparation

- CO will provide task and solutions in advance.
- **Try** the task yourself **without looking at solutions**.
  - Anticipate different ways that students might think about the task, and where they might get stuck.
  - Make notes of any useful bits of lecture notes, software documentation, or previous weeks' exercises to refer to.
- Ideally, keep the model solution only to check that results are correct. Students come up with lots of creative ways to solve a task – meet them where they are!

Helping students during labs

Helping students during labs

Same principles as for maths workshops:

- Don't give away the solution.
- Ask students to explain their thinking to you and to each other.
- Give pointers to course materials, documentation, etc.

Common questions and problems

- "I don't know where to start..."
- "What does this function do?"
- "Is this correct/will this work?"
- "This is not working."

Common questions and problems

- "I don't know where to start..."
- "What does this function do?"
- "Is this correct/will this work?"
- "This is not working."

Two of these questions are easy to answer...

Common questions and problems

- "I don't know where to start..."
- "What does this function do?" I don't know, let's find out!
- "Is this correct/will this work?"
- "This is not working."

Two of these questions are easy to answer...

Common questions and problems

- "I don't know where to start..."
- "What does this function do?" I don't know, let's find out!
- "Is this correct/will this work?" I don't know, let's find out!
- "This is not working."

Two of these questions are easy to answer...

I don't know where to start!

Translating problem to code is a skill that needs practice!

A useful guide: the Seven Steps method.

- Get some **pen and paper**.
- Work out **an example by hand**.
- **Retrace your steps**: write down exactly what you've done.
- **Generalise** the steps above to arbitrary values.
- **Test** your procedure on a different example.
- Then, and only then, translate to code.
- Test your programme on more examples.

I don't know where to start!

Translating problem to code is a skill that needs practice!

A useful guide: the Seven Steps method.

- Get some **pen and paper**.
- Work out **an example by hand**.
- **Retrace your steps**: write down exactly what you've done.
- **Generalise** the steps above to arbitrary values.
- **Test** your procedure on a different example.
- Then, and only then, translate to code.
- Test your programme on more examples.

I don't know where to start!

Translating problem to code is a skill that needs practice!

A useful guide: the Seven Steps method.

- Get some **pen and paper**.
- Work out **an example by hand**.
- **Retrace your steps**: write down exactly what you've done.
- **Generalise** the steps above to arbitrary values.
- **Test** your procedure on a different example.
- Then, and only then, translate to code.
- Test your programme on more examples.

I don't know where to start!

Translating problem to code is a skill that needs practice!

A useful guide: the Seven Steps method.

- Get some **pen and paper**.
- Work out **an example by hand**.
- **Retrace your steps**: write down exactly what you've done.
- **Generalise** the steps above to arbitrary values.
- **Test** your procedure on a different example.
- Then, and only then, translate to code.
- Test your programme on more examples.

I don't know where to start!

Translating problem to code is a skill that needs practice!

A useful guide: the Seven Steps method.

- Get some **pen and paper**.
- Work out **an example by hand**.
- **Retrace your steps**: write down exactly what you've done.
- **Generalise** the steps above to arbitrary values.
- **Test** your procedure on a different example.
- Then, and only then, translate to code.
- Test your programme on more examples.

I don't know where to start!

Translating problem to code is a skill that needs practice!

A useful guide: the Seven Steps method.

- Get some **pen and paper**.
- Work out **an example by hand**.
- **Retrace your steps**: write down exactly what you've done.
- **Generalise** the steps above to arbitrary values.
- **Test** your procedure on a different example.
- Then, and only then, translate to code.
- Test your programme on more examples.

I don't know where to start!

Translating problem to code is a skill that needs practice!

A useful guide: the Seven Steps method.

- Get some **pen and paper**.
- Work out **an example by hand**.
- **Retrace your steps**: write down exactly what you've done.
- **Generalise** the steps above to arbitrary values.
- **Test** your procedure on a different example.
- Then, and only then, translate to code.
- Test your programme on more examples.

I don't know where to start!

Translating problem to code is a skill that needs practice!

A useful guide: the Seven Steps method.

- Get some **pen and paper**.
- Work out **an example by hand**.
- **Retrace your steps**: write down exactly what you've done.
- **Generalise** the steps above to arbitrary values.
- **Test** your procedure on a different example.
- Then, and only then, translate to code.
- Test your programme on more examples.

This is not working. . .

Teach a man to fish. . .

Tutoring is **not** debugging students' code for them – it's helping them to develop the right habits to troubleshoot their own problems.

Computer labs can be the only place and time for students to learn to find and fix bugs, with guidance and support from tutors.

Resist the temptation to take the keyboard away from a student!

- Student won't remember what you've done.
- Very easy to use keyboard shortcuts without the student noticing.
- Very easy to skip explaining steps because we are used to them.
- *Very easy to accidentally convince a student that debugging requires expert knowledge that they do not have.*

This is not working. . .

Teach a man to fish. . .

Tutoring is **not** debugging students' code for them – it's helping them to develop the right habits to troubleshoot their own problems.

Computer labs can be the only place and time for students to learn to find and fix bugs, with guidance and support from tutors.

Resist the temptation to take the keyboard away from a student!

- Student won't remember what you've done.
- Very easy to use keyboard shortcuts without the student noticing.
- Very easy to skip explaining steps because we are used to them.
- *Very easy to accidentally convince a student that debugging requires expert knowledge that they do not have.*

This is not working. . .

Even though they provide incredibly useful information, novice coders are often fearful of **error messages**.

- As soon as a student comes to you with a runtime error: "let's look at the error message."
- Help them interpret the information there:
  - **Where** is the error in your code?
  - What **type** of error is this? Do you remember seeing an error like this before?
  - Googling an obscure error message can be helpful!

Demystifying errors is important to give students confidence. It's absolutely fine (I'd even say, encouraged) to say "I don't know", to ask for help from other tutors or students in the room, to run broken code, to spend a lot of time tracking down a bug with a student.

This is not working. . .

Even though they provide incredibly useful information, novice coders are often fearful of **error messages**.

- As soon as a student comes to you with a runtime error: "let's look at the error message."
- Help them interpret the information there:
  - **Where** is the error in your code?
  - What **type** of error is this? Do you remember seeing an error like this before?
  - Googling an obscure error message can be helpful!

Demystifying errors is important to give students confidence. It's absolutely fine (I'd even say, encouraged) to say "I don't know", to ask for help from other tutors or students in the room, to run broken code, to spend a lot of time tracking down a bug with a student.

This is not working. . .

Even though they provide incredibly useful information, novice coders are often fearful of **error messages**.

- As soon as a student comes to you with a runtime error: "let's look at the error message."
- Help them interpret the information there:
  - **Where** is the error in your code?
  - What **type** of error is this? Do you remember seeing an error like this before?
  - Googling an obscure error message can be helpful!

Demystifying errors is important to give students confidence. It's absolutely fine (I'd even say, encouraged) to say "I don't know", to ask for help from other tutors or students in the room, to run broken code, to spend a lot of time tracking down a bug with a student.

This is not working. . .

Even though they provide incredibly useful information, novice coders are often fearful of **error messages**.

- As soon as a student comes to you with a runtime error: "let's look at the error message."
- Help them interpret the information there:
  - **Where** is the error in your code?
  - What **type** of error is this? Do you remember seeing an error like this before?
  - Googling an obscure error message can be helpful!

Demystifying errors is important to give students confidence. It's absolutely fine (I'd even say, encouraged) to say "I don't know", to ask for help from other tutors or students in the room, to run broken code, to spend a lot of time tracking down a bug with a student.

This is not working. . .

Even though they provide incredibly useful information, novice coders are often fearful of **error messages**.

- As soon as a student comes to you with a runtime error: "let's look at the error message."
- Help them interpret the information there:
  - **Where** is the error in your code?
  - What **type** of error is this? Do you remember seeing an error like this before?
  - Googling an obscure error message can be helpful!

Demystifying errors is important to give students confidence. It's absolutely fine (I'd even say, encouraged) to say "I don't know", to ask for help from other tutors or students in the room, to run broken code, to spend a lot of time tracking down a bug with a student.

This is not working. . .

Even though they provide incredibly useful information, novice coders are often fearful of **error messages**.

- As soon as a student comes to you with a runtime error: "let's look at the error message."
- Help them interpret the information there:
  - **Where** is the error in your code?
  - What **type** of error is this? Do you remember seeing an error like this before?
  - Googling an obscure error message can be helpful!

Demystifying errors is important to give students confidence. It's absolutely fine (I'd even say, encouraged) to say "I don't know", to ask for help from other tutors or students in the room, to run broken code, to spend a lot of time tracking down a bug with a student.

This is not working...

Even though they provide incredibly useful information, novice coders are often fearful of **error messages**.

- As soon as a student comes to you with a runtime error: "let's look at the error message."
- Help them interpret the information there:
  - **Where** is the error in your code?
  - What **type** of error is this? Do you remember seeing an error like this before?
  - Googling an obscure error message can be helpful!

Demystifying errors is important to give students confidence. It's absolutely fine (I'd even say, encouraged) to say "I don't know", to ask for help from other tutors or students in the room, to run broken code, to spend a lot of time tracking down a bug with a student.

Practical strategies: the rubber duck

For structure or logical issues: Rubber duck debugging

Become the rubber duck.

Ask students to explain to you, **line by line, in excruciating detail,** what their code is doing.

In the large majority of cases, they will find their mistake as they're explaining it.

As a tutor, you can be a slightly more active rubber duck.

- **Ask prompting questions** to help students through the explanation. (Are you sure? How does that function work? How many times do you do this? Let's try an example; etc.)
- Encourage them to **Google things**, and to reuse code snippets responsibly. (Ask me later about citing code appropriately!)
- Encourage them to **display** intermediate values, or plot results, to check that their code is actually doing what they intend.

Practical strategies: the rubber duck

For structure or logical issues: Rubber duck debugging

Become the rubber duck.

Ask students to explain to you, **line by line, in excruciating detail,** what their code is doing.

In the large majority of cases, they will find their mistake as they're explaining it.

As a tutor, you can be a slightly more active rubber duck.

- **Ask prompting questions** to help students through the explanation. (Are you sure? How does that function work? How many times do you do this? Let's try an example; etc.)
- Encourage them to **Google things**, and to reuse code snippets responsibly. (Ask me later about citing code appropriately!)
- Encourage them to **display** intermediate values, or plot results, to check that their code is actually doing what they intend.

Practical strategies: the rubber duck

For structure or logical issues: Rubber duck debugging

Become the rubber duck.

Ask students to explain to you, **line by line, in excruciating detail,** what their code is doing.

In the large majority of cases, they will find their mistake as they're explaining it.

As a tutor, you can be a slightly more active rubber duck.

- **Ask prompting questions** to help students through the explanation. (Are you sure? How does that function work? How many times do you do this? Let's try an example; etc.)
- Encourage them to **Google things**, and to reuse code snippets responsibly. (Ask me later about citing code appropriately!)
- Encourage them to **display** intermediate values, or plot results, to check that their code is actually doing what they intend.

Practical strategies: the rubber duck

For structure or logical issues: Rubber duck debugging

Become the rubber duck.

Ask students to explain to you, **line by line, in excruciating detail,** what their code is doing.

In the large majority of cases, they will find their mistake as they're explaining it.

As a tutor, you can be a slightly more active rubber duck.

- **Ask prompting questions** to help students through the explanation. (Are you sure? How does that function work? How many times do you do this? Let's try an example; etc.)
- Encourage them to **Google things**, and to reuse code snippets responsibly. (Ask me later about citing code appropriately!)
- Encourage them to **display** intermediate values, or plot results, to check that their code is actually doing what they intend.

Practical strategies: the rubber duck

For structure or logical issues: Rubber duck debugging

Become the rubber duck.

Ask students to explain to you, **line by line, in excruciating detail,** what their code is doing.

In the large majority of cases, they will find their mistake as they're explaining it.

As a tutor, you can be a slightly more active rubber duck.

- **Ask prompting questions** to help students through the explanation. (Are you sure? How does that function work? How many times do you do this? Let's try an example; etc.)
- Encourage them to **Google things**, and to reuse code snippets responsibly. (Ask me later about citing code appropriately!)
- Encourage them to **display** intermediate values, or plot results, to check that their code is actually doing what they intend.

Putting it into practice

Resources

Resources

- The Seven Steps method: poster (linked above) and accompanying paper with further details.
- The Teach Computing project has a good collection of resources. Highlights on 2 resources to support program comprehension:
  - Code tracing
  - The block model
- Brown, N., & Wilson, G. (2018). Ten quick tips for teaching programming. PLoS computational biology, 14(4), e1006023. https://doi.org/10.1371/journal.pcbi.1006023
- The Computing Education Research Blog by Prof. Mark Guzdial at the University of Michigan.
- Software Carpentry's instructor training material. Includes evidence-based, practical advice on supporting students in learning computing.