

About this Site

Contents

Syllabus

- [Syllabus](#)
- [Basic Facts](#)
- [Introduction to Computer Systems](#)
- [Tools and Resources](#)
- [Schedule](#)
- [Grading](#)
- [Grading Policies](#)
- [Support](#)
- [General URI Policies](#)
- [Course Communications](#)

Notes

- [1. Introduction](#)
- [2. How does learning and knowledge work in computing?](#)

FAQ

- [Syllabus and Grading FAQ](#)
- [Git and GitHub](#)

Resources

- [Glossary](#)
- [Language Specific References](#)
- [Cheatsheet](#)
- [General Tips and Resources](#)
- [How to Study in this class](#)
- [Getting Help with Programming](#)
- [Getting Organized for class](#)
- [Advice from Dr. Brown's Data Science Students](#)

Welcome to the course manual for Introduction to Computer Systems in with Professor Brown.

This class meets TuTH 12:30-1:45 in Engineering Building Room 040.

This website will contain the syllabus, class notes, and other reference material for the class.

[Course Calendar on BrightSpace](#)



Tip

[subscribe to that calendar](#) in your favorite calendar application

Navigating the Sections

The Syllabus section has logistical operations for the course broken down into sections. You can also read straight through by starting in the first one and navigating to the next section using the arrow navigation at the end of the page.

This site is a resource for the course. We do not follow a text book for this course, but all notes from class are posted in the notes section, accessible on the left hand side menu, visible on large screens and in the menu on mobile.

The resources section has links and short posts that provide more context and explanation. Content in this section is for the most part not strictly the material that you'll be graded on, but it is often material that will help you understand and grow as a programmer and data scientist.

Reading each page

All class notes can be downloaded in multiple formats, including as a notebook. Some pages of the syllabus and resources are also notebooks, if you want to see behind the curtain of how I manage the course information.

Try it Yourself

Notes will have exercises marked like this

Question from Class

Questions that are asked in class, but unanswered at that time will be answered in the notes and marked with a box like this. Long answers will be in the main notes

Further reading

Notes that are mostly links to background and context will be highlighted like this. These are optional, but will mostly help you understand code excerpts they relate to.

Question from class

Questions that are asked in class, but unanswered at that time will be answered in the notes and marked with a box like this. Short questions will be in the margin note

Hint

Both notes and assignment pages will have hints from time to time. Pay attention to these on the notes, they'll typically relate to things that will appear in the assignment.

Think Ahead

Think ahead boxes will guide you to start thinking about what can go into your portfolio to build on the material at hand.

Ram Token Opportunity

Chances to earn ram tokens are highlighted this way.

Syllabus

Welcome to CSC302: Introduction to Computer Systems.

In this syllabus you will find an overview of the course, information about your instructor, course policies, restatements of URI policies, reminders of relevant resources, and a schedule for the course.

This is a live document that will change over time, but a pdf copy is available for direct [download](#) or to [view on GitHub](#). Note that this will become outdated over time.

Basic Facts

Introduction to Computer Systems

This new course links together different ideas that you have encountered but not covered deeply in other courses. We'll learn about tools used in programming and how they work. The goal of this course is to help you understand how your computer and programming environment work so that you can debug and learn independently more confident.

Quick Facts

- **Course time:** Spring 2022, TuTh 12:30PM - 1:45PM
- **Credits:** 4

To request a permission number [complete this google form](#) you must be signed into your URI google account to access the form

Why Take this course

1. use and understand git/ GitHub
2. make sense of cryptic compiler messages
3. understand how file organization impacts programming
4. fulfill your 300 level CSC elective requirement
5. preview ideas that will be explored in depth in 411 & 412

Topics covered

this is a partial list

- git and other version control
- bash and other shell scripting
- filesystems
- basics of hardware
- what happens when you compile code
- what are the different types of software on your computer

Catalog Description

How the history and context of computing impacts the practice of computing today. Tools used in programming and computational problem solving. How programming works from high level languages to hardware. Survey of computer hardware and representation of information. Pre: CSC110, any 200 level CSC course, or equivalent.

Learning Outcomes

By the end of the semester, students will be able to:

1. Differentiate the different classes of tools used in computer science in terms of their features, roles, and how they interact and justify positions and preferences among popular tools
2. Identify the computational pipeline from hardware to high level programming language
3. Discuss implications of choices across levels of abstraction

4. Describe the context under which essential components of computing systems were developed and explain the impact of that context on the systems.

About this syllabus

You can get notification of changes from GitHub by “watching” the [repository](#). You can view the date of changes and exactly what changes were made on the Github [commit history](#) page.

Creating an [issue](#) is also a good way to ask questions about anything in the course it will prompt additions and expand the FAQ section. That will be linked when solved and you will get a notification at that time.

About your instructor

Name: Dr. Sarah M Brown Office hours: TBA via zoom, link on BrightSpace

Dr. Sarah M Brown is a second year Assistant Professor of Computer Science, who does research on how social context changes machine learning. Dr. Brown earned a PhD in Electrical Engineering from Northeastern University, completed a postdoctoral fellowship at University of California Berkeley, and worked as a postdoctoral research associate at Brown University before joining URI. At Brown University, Dr. Brown taught the Data and Society course for the Master's in Data Science Program. You can learn more about me at my [website](#) or my research on my [lab site](#).

You can call me Professor Brown or Dr. Brown, I use she/her pronouns.

The best way to contact me is e-mail or an issue on an assignment repo. For more details, see the [Communication Section](#)

Tools and Resources

We will use a variety of tools to conduct class and to facilitate your programming. You will need a computer with Linux, MacOS, or Windows. It is unlikely that a tablet will be able to do all of the things required in this course. A Chromebook may work, especially with developer tools turned on. Ask Dr. Brown if you need help getting access to an adequate computer.

All of the tools and resources below are either:

- paid for by URI **OR**
- freely available online.

BrightSpace

Note

Seeing the BrightSpace site requires logging in with your URI SSO and being enrolled in the course

This will be the central location from which you can access all other materials. Any links that are for private discussion among those enrolled in the course will be available only from our course [Brightspace site](#).

This is also where your grades will appear and how I will post announcements.

For announcements, you can [customize](#) how you receive them.

Prismia chat

Our class link for [Prismia chat](#) is available on Brightspace. Once you've joined once, you can use the link above or type the url: prismia.chat. We will use this for chatting and in-class understanding checks.

On Prismia, all students see the instructor's messages, but only the Instructor and TA see student responses.

Course Manual

The course manual will have content including the class policies, scheduling, class notes, assignment information, and additional resources.

Links to the course reference text and code documentation will also be included here in the assignments and class notes.

GitHub

You will need a [GitHub](#) Account. If you do not already have one, please [create one](#) by the first day of class. If you have one, but have not used it recently, you may need to update your password and login credentials as the [Authentication rules](#) changed in Summer 2021. In order to use the command line with https, you will need to [create a Personal Access Token](#) for each device you use. In order to use the command line with SSH, set up your public key.

Programming Environment

In this course, we will use several programming environments. In order to complete assignments you need the items listed in the requirements list. The easiest way to meet these requirements is to follow the recommendations below. I will provide instruction assuming that you have followed the recommendations. We will add tools throughout the semester, but the following will be enough to get started.

Warning

This is not technically a *programming* class, so you will not need to know how to write code from scratch in specific languages, but we will rely on programming environments to apply concepts.

Requirements:

- Python with scientific computing packages (numpy, scipy, jupyter, pandas, seaborn, sklearn)
- [Git](#)
- A bash shell
- A web browser compatible with [Jupyter Notebooks](#)
- nano text editor

Note

all Git instructions will be given as instructions for the command line interface and GitHub specific instructions via the web interface. You may choose to use GitHub desktop or built in IDE tools, but the instructional team may not be able to help.

Warning

Everything in this class will be tested with the up to date (or otherwise specified) version of Jupyter Notebooks. Google Colab is similar, but not the same, and some things may not work there. It is an okay backup, but should not be your primary work environment.

Recommendation:

- Install python via [Anaconda](#)
- if you use Windows, install Git and Bash with [GitBash](#) ([video instructions](#)).
- if you use MacOS, install Git with the Xcode Command Line Tools. On Mavericks (10.9) or above you can do this by trying to run git from the Terminal the very first time.`git --version`
- if you use Chrome OS, follow these instructions:

1. Find Linux (Beta) in your settings and turn that on.
2. Once the download finishes a Linux terminal will open, then enter the commands: `sudo apt-get update` and `sudo apt-get upgrade`. These commands will ensure you are up to date.
3. Install tmux with:

```
sudo apt -t stretch-backports install tmux
```

4. Next you will install nodejs, to do this, use the following commands:

```
curl -sL https://deb.nodesource.com/setup_14.x | sudo -E bash  
sudo apt-get install -y nodejs  
sudo apt-get install -y build-essential.
```

5. Next install Anaconda's Python from the website provided by the instructor and use the top download link under the Linux options.
6. You will then see a .sh file in your downloads, move this into your Linux files.
7. Make sure you are in your home directory (something like home/YOURUSERNAME), do this by using the `pwd` command.
8. Use the `bash` command followed by the file name of the installer you just downloaded to start the installation.
9. Next you will add Anaconda to your Linux PATH, do this by using the `vim .bashrc` command to enter the .bashrc file, then add the `export PATH=/home/YOURUSERNAME/anaconda3/bin/:$PATH` line. This can be placed at the end of the file.
10. Once that is inserted you may close and save the file, to do this hold escape and type `:x`, then press enter. After doing that you will be returned to the terminal where you will then type the `source .bashrc` command.
11. Next, use the `jupyter notebook --generate-config` command to generate a Jupyter Notebook.
12. Then just type `jupyter lab` and a Jupyter Notebook should open up.

Video install instructions for Anaconda:

- [Windows](#)
- [Mac](#)

On Mac, to install python via environment, [this article may be helpful](#)

- I don't have a video for linux, but it's a little more straight forward.

Zoom (backup only & office hours only, Spring 2022 is in person)

This is where we will meet if for any reason we cannot be in person. You will find the link to class zoom sessions on Brightspace.

URI provides all faculty, staff, and students with a paid Zoom account. It *can* run in your browser or on a mobile device, but you will be able to participate in class best if you download the [Zoom client](#) on your computer. Please [log in](#) and [configure your account](#). Please add a photo of yourself to your account so that we can still see your likeness in some form when your camera is off. You may also wish to use a virtual background and you are welcome to do so.

For help, you can access the [instructions provided by IT](#).

Schedule

Overview

The following is a rough outline of topics in an order, these things will be filled into the concrete schedule above as we go. These are, in most cases bigger questions than we can tackle in one class, but will give the general idea of how the class will go.

This plan accounts for 1 less week than we actually have. We will either go over somewhere or we'll use the last week for sharing projects, reflection, or an additional topic that comes up during the semester.

How does this class work?

one week

We'll spend the first two classes introducing some basics of GitHub and setting expectations for how the course will work. This will include how you are expected to learn in this class which requires a bit about how knowledge production in computer science works and a bit of the history.

How do all of these topics relate?

approximatley two weeks

Tip

We will integrate history throughout the whole course. Connecting ideas to one another, and especially in a sort of narrative form can help improve retention of ideas. My goal is for you to learn.

We'll also come back to different topics over and over again with a slightly different framing each time. This will both connect ideas, give you chance to practice recalling (more recall practice improves long term retention of things you learn), and give you a chance to learn things in different ways.

We'll spend a few classes doing an overview where we go through each topic in a litte more depth than an introduction, but not as deep as the rest of the semester. In this section, we will focus on how the different things we will see later all relate to one another more than a deep understanding of each one. At the end of this unit, we'll work on your grading contracts.

We'll also learn more key points in history of computing to help tie concepts together in a narrative.

Topics:

- bash
- man pages (built in help)
- terminal text editor
- git
- survey of hardware
- compilation
- information vs data

What tools do Computer Scientists use?

approximately four weeks

Next we'll focus in on tools we use as computer scientists to do our work. We will use this as a way to motivate how different aspects of a computer work in greater detail.

Topics:

- linux
- git
- i/o
- ssh and ssh keys
- number systems
- file systems

What Happens When I run code?

approximately five weeks

Finally, we'll go in really deep on the compilation and running of code. In this part, we will work from the compilation through to assembly down to hardware and then into machine representation of data.

Topics:

- software system and Abstraction
- programming languages
- cache and memory
- compilation
- linking
- basic hardware components

Finalized Order

Content from above will be expanded and slotted into specific classes as we go. This will always be a place you can get reminders of what you need to do next and/or what you missed if you miss a class as an overivew. More Details will be in other parts of the site, linked to here.

Date	Key Question	Prepation	Activities
2021-01-25	What are we doing this semester?	Create GitHub and Prismia accounts, take stock of dev environments	introductions, tool practice
2021-01-27	How does knowledge work in computing?	Read through the class site, notes, reflect on a thing you know well	course FAQ, knowledge discussion
2021-02-01	How do I use git offline?	review notes, reflect on issues, check environment, map cs knowledge	cloning, pushing, terminal basics
2021-02-03			
2021-02-08			
2021-02-10			
2021-02-15			
2021-02-17			
2021-02-22			
2021-02-24			
2021-03-01			
2021-03-03			
2021-03-08			
2021-03-10			
2021-03-22			
2021-03-24			
2021-03-29			
2021-03-31			
2021-04-05			
2021-04-07			
2021-04-12			
2021-04-14			
2021-04-19			
2021-04-21			
2021-04-26			
2021-04-28			

Table 1 Schedule

Grading

This section of the syllabus describes the principles and mechanics of the grading for the course.

Learning Outcomes

The goal is for you to learn and the grading is designed to as close as possible actually align to how much you have learned. So, the first thing to keep in mind, always is the course learning outcomes:

By the end of the semester, students will be able to:

1. Differentiate the different classes of tools used in computer science in terms of their features, roles, and how they interact and justify positions and preferences among popular tools
2. Identify the computational pipeline from hardware to high level programming language
3. Discuss implications of choices across levels of abstraction
4. Describe the context under which essential components of computing systems were developed and explain the impact of that context on the systems.

These are what I will be looking for evidence of to say that you met those or not.

Principles of Grading

Learning happens through practice and feedback. My goal as a teacher is for you to learn. The grading in this course is based on your learning of the material, rather than your completion of the activities that are assigned.

This course is designed to encourage you to work steadily at learning the material and demonstrating your new knowledge. There are no single points of failure, where you lose points that cannot be recovered. Also, you cannot cram anything one time and then forget it. The material will build and you have to demonstrate that you retained things.

- Earning a C in this class means you have a general understanding; you will know what all the terms mean and could follow along if in a meeting where others were discussing systems concepts.
- Earning a B means that you could apply the course concepts in other programming environments; you can solve basic common errors without looking much up.
- Earning an A means that you can use knowledge from this course to debug tricky scenarios and/or design aspects of systems; you can solve uncommon error while only looking up specific syntax, but you have an idea of where to start.

No Grade Zone

At the beginning of the course we will have a grade free zone where you practice with both course concepts and the tooling and assignment types to get used to expectations. You will get feedback on lots of work and begin your Know, Want to know, Learned (KWL) Chart in this period.

Grading Contract

In about the third week you will complete, from a provided template, a grading contract. In that you will state what grade you want to earn in the class and what work you are going to do to show that. If you complete all of that work to a satisfactory level, you will get that grade. The grade free zone is a chance for you to get used to the type of feedback in the course and the grading contract template will have example specifications to meet.

The finalized grading contract will include the specification that each piece of work has to adhere to.

All contracts will include maintaining a KWL Chart for the duration of the semester and consistent responses in class.

Grading Policies

Late Work

You will get feedback on items at the next feedback period.

Regrading

Re-request a review on your Feedback Pull request.

For general questions, post on the conversation tab of your Feedback PR with your request.

For specific questions, reply to a specific comment.

If you think we missed *where* you did something, add a comment on that line (on the code tab of the PR, click the plus (+) next to the line) and then post on the conversation tab with an overview of what you're requesting and tag @brownsarahm

Support

Academic Enhancement Center

Academic Enhancement Center (for undergraduate courses): Located in Roosevelt Hall, the AEC offers free face-to-face and web-based services to undergraduate students seeking academic support. Peer tutoring is available for STEM-related courses by appointment online and in-person. The Writing Center offers peer tutoring focused on supporting undergraduate writers at any stage of a writing assignment. The UCS160 course and academic skills consultations offer students strategies and activities aimed at improving their studying and test-taking skills. Complete details about each of these programs, up-to-date schedules, contact information and self-service study resources are all available on the [AEC website](#).

- **STEM Tutoring** helps students navigate 100 and 200 level math, chemistry, physics, biology, and other select STEM courses. The STEM Tutoring program offers free online and limited in-person peer-tutoring this fall. Undergraduates in introductory STEM courses have a variety of small group times to choose from and can select occasional or weekly appointments. Appointments and locations will be visible in the TutorTrac system on September 14th, 2020. The TutorTrac application is available through [URI Microsoft 365 single sign-on](#) and by visiting [aec.uri.edu](#). More detailed information and instructions can be found on the [AEC tutoring page](#).
- **Academic Skills Development** resources helps students plan work, manage time, and study more effectively. In Fall 2020, all Academic Skills and Strategies programming are offered both online and in-person. UCS160: Success in Higher Education is a one-credit course on developing a more effective approach to studying. Academic Consultations are 30-minute, 1 to 1 appointments that students can schedule on Starfish with Dr. David Hayes to address individual academic issues. Study Your Way to Success is a self-guided web portal connecting students to tips and strategies on studying and time management related topics. For more information on these programs, visit the [Academic Skills Page](#) or contact Dr. Hayes directly at davidhayes@uri.edu.
- The **Undergraduate Writing Center** provides free writing support to students in any class, at any stage of the writing process: from understanding an assignment and brainstorming ideas, to developing, organizing, and revising a draft. Fall 2020 services are offered through two online options: 1) real-time synchronous appointments with a peer consultant (25- and 50-minute slots, available Sunday - Friday), and 2) written asynchronous consultations with a 24-hour turn-around response time (available Monday - Friday). Synchronous appointments are video-based, with audio, chat, document-sharing, and live captioning capabilities, to meet a range of accessibility needs. View the synchronous and asynchronous schedules and book online, visit [uri.mywconline.com](#).

General URI Policies

Anti-Bias Statement:

We respect the rights and dignity of each individual and group. We reject prejudice and intolerance, and we work to understand differences. We believe that equity and inclusion are critical components for campus community members to thrive. If you are a target or a witness of a bias incident, you are encouraged to submit a report to the URI Bias Response Team at www.uri.edu/brt. There you will also find people and resources to help.

Disability Services for Students Statement:

Your access in this course is important. Please send me your Disability Services for Students (DSS) accommodation letter early in the semester so that we have adequate time to discuss and arrange your approved academic accommodations. If you have not yet established services through DSS, please contact them to engage in a confidential conversation about the process for requesting reasonable accommodations in the classroom. DSS can be reached by calling: 401-874-2098, visiting: web.uri.edu/disability, or emailing: dss@etal.uri.edu. We are available to meet with students enrolled in Kingston as well as Providence courses.

Academic Honesty

Students are expected to be honest in all academic work. A student's name on any written work, quiz or exam shall be regarded as assurance that the work is the result of the student's own independent thought and study. Work should be stated in the student's own words, properly attributed to its source. Students have an obligation to know how to quote, paraphrase, summarize, cite and reference the work of others with integrity. The following are examples of academic dishonesty.

- Using material, directly or paraphrasing, from published sources (print or electronic) without appropriate citation
- Claiming disproportionate credit for work not done independently
- Unauthorized possession or access to exams
- Unauthorized communication during exams
- Unauthorized use of another's work or preparing work for another student
- Taking an exam for another student
- Altering or attempting to alter grades
- The use of notes or electronic devices to gain an unauthorized advantage during exams
- Fabricating or falsifying facts, data or references
- Facilitating or aiding another's academic dishonesty
- Submitting the same paper for more than one course without prior approval from the instructors

URI COVID-19 Statement

The University is committed to delivering its educational mission while protecting the health and safety of our community. While the university has worked to create a healthy learning environment for all, it is up to all of us to ensure our campus stays that way.

As members of the URI community, students are required to comply with standards of conduct and take precautions to keep themselves and others safe. Visit web.uri.edu/coronavirus/ for the latest information about the URI COVID-19 response.

- [Universal indoor masking](#) is required by all community members, on all campuses, regardless of vaccination status. If the universal mask mandate is discontinued during the semester, students who have an approved exemption and are not fully vaccinated will need to continue to wear a mask indoors and maintain physical distance.
- Students who are experiencing symptoms of illness should not come to class. Please stay in your home/room and notify URI Health Services via phone at 401-874-2246.
- If you are already on campus and start to feel ill, go home/back to your room and self-isolate. Notify URI Health Services via phone immediately at 401-874-2246.

If you are unable to attend class, please notify me at brownsarahm@uri.edu. We will work together to ensure that course instruction and work is completed for the semester.

Course Communications

Help Hours

TBA

Day	Time	Location	Host
Tuesday	4-5pm	online	Dr. Brown
Wednesday	1-2pm	online	Dr. Brown
Friday	11am-1pm	online	Mark


Online office hours locations are linked in the #help channel on slack

Tips

For assignment help

- **send in advance, leave time for a response** I check e-mail/github a small number of times per day, during work hours, almost exclusively. You might see me post to this site, post to BrightSpace, or comment on your assignments outside of my normal working hours, but I will not reliably see emails that arrive during those hours. This means that it is important to start assignments early.

Using issues

- use issues for content directly related to assignments. If you push your code to the repository and then open an issue, I can see your code and your question at the same time and download it to run it if I need to debug it
- use issues for questions about this syllabus or class notes. At the top right there's a GitHub logo  that allows you to open an issue (for a question) or suggest an edit (eg if you think there's a typo or you find an additional helpful resource related to something)

For E-mail

- use e-mail for general inquiries or notifications
- Please include **[CSC392]** in the subject line of your email along with the topic of your message. This is important, because your messages are important, but I also get a lot of e-mail. Consider these a cheat code to my inbox: I have setup a filter that will flag your e-mail if you include that in subject to ensure that I see it.

1. Introduction

1.1. What is a System?

💡 Tip

You can contribute or fix things on this page (and anywhere else in this site) by clicking on "suggest an edit" under the GitHub menu in the top right.

1.2. What are we going to learn? and Editing on GitHub

We initialized your [KWL Chart](#). You will keep this chart up to date over the course of the semester. Mostly it will be prompted when you should fill it in, but you can add to it whenever you would like.

📖 Further Reading

GitHub itself provides pretty good documentation, full of screenshots for things in browser. [editing a file pull request](#)

1.3. For next class

📌 Note

This section will contain more detail, but a short list of what you need will always be in the [schedule](#)

- More practice with [GitHub terminology](#). Accept this assignment, read through it, and follow the instructions at the end.
- Review these notes, bring any questions you have to class
- Read the syllabus, explore this whole [website](#). Bring questions about the course. Be prepared for a scavenger hunt that asks you not to recall every fact about the course, but to know where to find information.
- Think about one thing you've learned really well (computing or not) and how do you know that you know it? (bring your example)

1.4. Questions After Class

1.4.1. What physical code will we be writing this semester?

1.4.2. How would committing, pull requests, branches, etc work if you wanted to work on something on your computer?

1.4.3. Why does github seem so simple on a surface level? but to actually use it requires much deeper knowledge...

1.4.4. Will I be able to view my answers on prismia chat for the future? Will it erase?

1.4.5. What about the feedback page? How to merge it or not merge it at all?

2. How does learning and knowledge work in computing?

2.1. Git review

- Make sure you get [GitHub terminology](#) down and use this "assignment" to practice.

2.2. Scavenger Hunt

Note

The goal here is to make sure you know where to find basic things, not that you have memorized every bit of information about the course

Where can you find when office hours are?



Where can you find the detailed list of what to prepare for today's class?



Where is the regrading policy?



There's a term you don't recognize in an activity, where should you look?



Something went wrong in an assignment repo on GitHub, what should you check before asking for help?



2.3. Recall, Systems

"Systems" in computing often refers to all the parts that help make the "more exciting" algorithmic parts work. Systems is like the magic that helps you get things done in practice, so that you can shift your attention elsewhere.

In intro courses, we typically give you an environment to hide all the problems that could occur at the systems level.

Systems programming is how to look at the file system, the operating system, etc.

2.4. Mental Models and Learning

2.4.1. What is it like to know something really well?

When we know something well, it is easier to do, we can do it multiple ways, it is easy to explain to others and we can explain it multiple ways. we can do the task almost automatically and combine and create things in new ways. This is true for all sorts of things.

a mental model is how you think about a concept and your way of relating it. Novices have sparse mental models, experts have connected mental models.

When we first learn new things, we first get the basic concepts down, but we may not know how they relate.

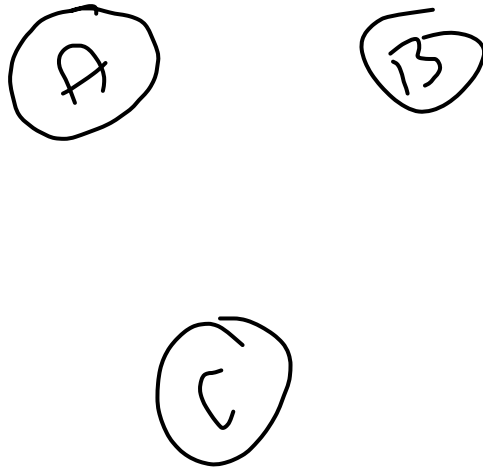


Fig. 2.1 a novice mental model is disconnected and has few concepts

As we learn more, they become more connected.

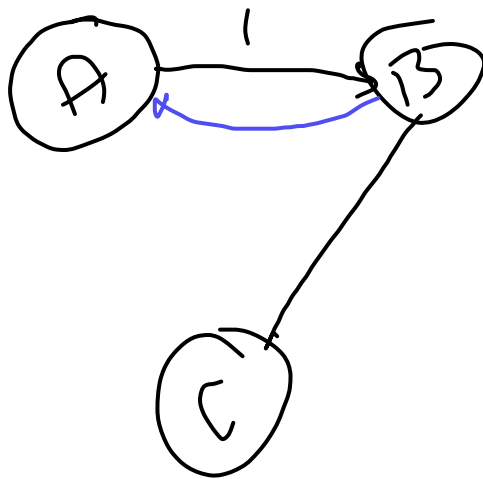


Fig. 2.2 a competent mental model starts to have some connections, with relationships between the concepts.

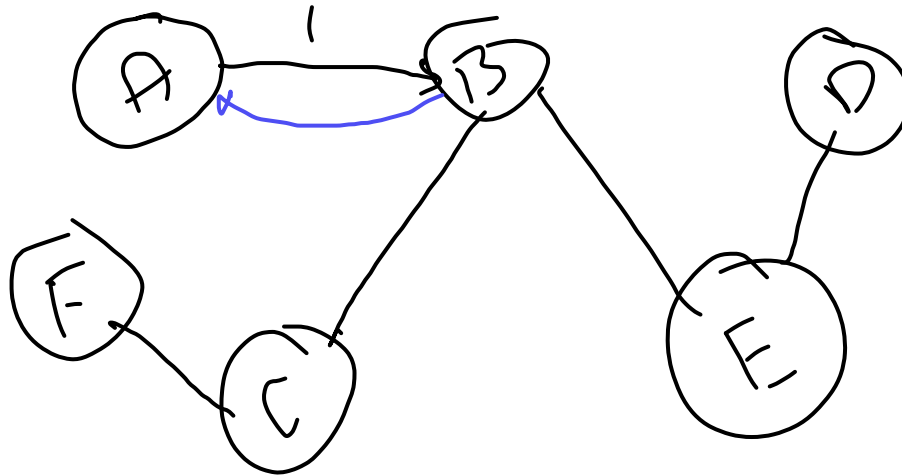


Fig. 2.3 an expert mental model is densely connected and has more concepts in it.

We can visualize with concept maps. Which connect the ideas using relationships on the arrows.

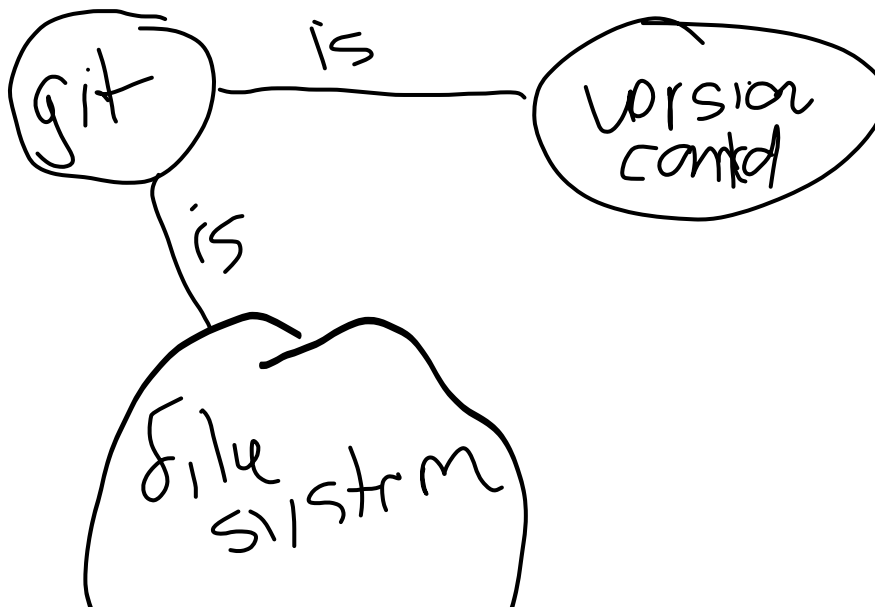


Fig. 2.4 a small concept map showing that git is an instance of both a file system and a version control system.

2.5. Why do we need this for computer systems?

Attention

This section contains points added here that were not discussed directly in class, but are important and will come back up

2.5.1. Systems are designed by programmers

Computer Science is not a natural science like biology or physics where we try to understand some aspect of the world that we live in. Computer Science as a discipline, like algorithms, mostly derives from Math.

Historically, Computer Science Departments were often initially formed by professors in math creating a new department or, sometimes, making a new degree programs without even creating a new department at first. In some places, CS degree programs also grew within or out of Electrical Engineering. At URI, CS grew out of math.

So, when we study computer science, while parts of it are limited by physics^[1], most of it is essentially an imaginary world that is made by people. Understanding how people think, both generally, and common patterns within the community of programmers^[2] understand how things work and why they are the way they are. The why can also make it easier to remember, or, it can help you know what things you can find alternatives for, or even where you might invent a whole new thing that is better in some way.

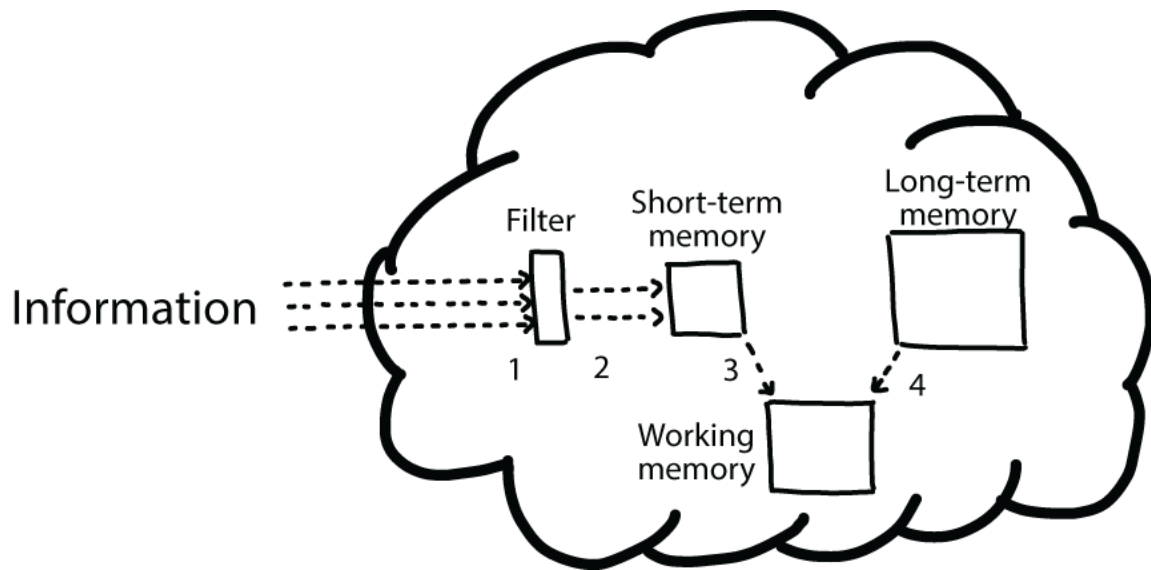


Fig. 2.5 An overview of the three cognitive processes that [this book](#) covers: STM, LTM, and working memory. The arrows labeled 1 represent information coming into your brain. The arrows labeled 2 indicate the information that proceeds into your STM. Arrow 3 represents information traveling from the STM into the working memory, where it's combined with information from the LTM (arrow 4). Working memory is where the information is processed while you think about it.

2.5.2. Context Matters

This context of how things were developed can influence how we understand it. We will also talk about the history of computing as we go through different topics in class so that we can build that context up.

2.5.3. Optimal is relative

The “best” way to do something is always relative to the context. “Best” is a vague term. It could be most computationally efficient theoretically, fastest to run on a particular type of hardware, or easiest for another programmer to read.

We will see how the best choice varies a lot as we investigate things at different levels of abstraction.

2.6. How I expect this to work

2.7. For next class

Note

This is what is required, before the next class and will be checked or if you don't do it you will have trouble participating in class

1. Review these notes, both rendered as html and the raw markdown in the repository.
2. find 2-3 examples of things in programming you have got working, but did not really understand. this could be errors you fixed, or something you just know you're supposed to do, but not why
3. map out your computing knowledge and add it to your kwl chart repo. this can be an image that you upload or a text-based outline.
4. Make sure you have a working environment for next week. Use slack to ask for help.
 - check that you have Python installed with Jupyter, ideally with [Anaconda](#)
 - install [jupyter book](#)
 - install [GitBash](#) on windows (optional for others)
 - make sure you have Xcode on MacOS
 - install [the GitHub CLI](#) on all OSs

2.8. More Practice

Note

Activities in this section are optional, but things that may help you prepare, or (in future classes) extend the idea.

- (optional) try mapping out using [mermaid](#) syntax, we'll be using other tools that will facilitate rendering later, or try getting it to render on your own.
- (optional) read chapter 1 [the programmer's brain](#). Some of the ideas we talked about today are mentioned there, and it relates to where you're supposed to be looking for things that you have done, but didn't really understand.
- try adding something to this page or the glossary of the course site or link a glossary term to an occurrence of it on the site.

Hint

terms on this page that could be added to the glossary include [filesystem](#) and [operating system](#). The [jupyter book docs](#) show how to add to a glossary and link to the glossary from another page.

2.9. Questions After Class

2.9.1. How would I learn more about version control systems?

2.9.2. How to use github to make more meaningful repositories, instead of just a mess of files that are not properly uploaded?

2.9.3. Are there any benefits to using git offline vs using it only in conjunction with github?

2.9.4. When will we be establishing the grade contracts?

2.9.5. How can I be better at communicating documentation

2.9.6. How often are we supposed to update our KWL Charts?

2.9.7. Do we get a notification when you post the notes, or do we just check periodically?

[1] when we are *really* close to the hardware

[2] Of course, not *all* programmers think the same way, but when people spend time together and communicate, they start to share patterns in how they think. So, while you do **not** have to think the same way as these patterns, knowing what they are will help you reading code, and understanding things.

Syllabus and Grading FAQ

Git and GitHub

I can't push to my repository, I get an error that updates were rejected

My command line says I cannot use a password

Help! I accidentally merged the Feedback Pull Request before my assignment was graded

Glossary

Tip

We will build a glossary as the semester goes on. When you encounter a term you do not know, create an issue to ask for help, or contribute a PR after you find the answer.

git

a version control tool; it's a fully open source and always free tool, that can be hosted by anyone or used without a host, locally only.

GitHub

a hosting service for git repositories

repository

a project folder with tracking information in it in the form of a .git file

Language Specific References

Python

- [Python](#)

Cheatsheet

Patterns and examples of how to accomplish frequent tasks. We will build up this section together over the course of the semester.

General Tips and Resources

This section is for materials that are not specific to this course, but are likely useful. They are not generally required readings or installs, but are options or advice I provide frequently.

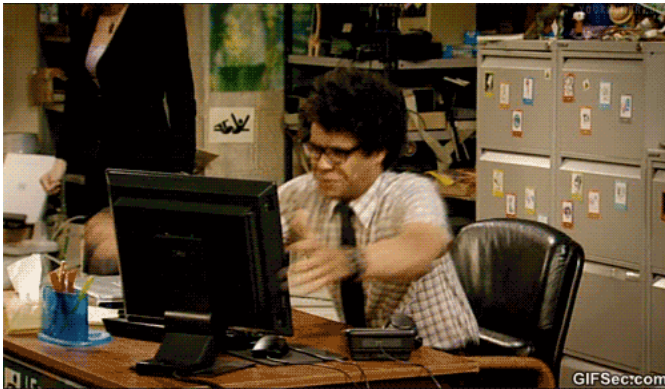
on email

- [how to e-mail professors](#)

How to Study in this class

In this page, I break down how I expect learning to work for this class.

I hope that with this advice, you never feel like this while working on assignments for this class.



Why this way?

A new book that might be of interest if you find programming classes hard is [the Programmers Brain](#). As of 2021-09-07, it is available for free by clicking on chapters at that linked table of contents section.

Learning requires iterative practice. It does not require memorizing all of the specific commands, but instead learning the basic patterns.

Using reference materials frequently is a built in part of programming, most languages have built in help as a part of the language for this reason. This course is designed to have you not only learn the material, but also to build skill in learning to program. Following these guidelines will help you build habits to not only be successful in this class, but also in future programming.

Learning in class

! Important

My goal is to use class time so that you can be successful with *minimal frustration* while working outside of class time.

Programming requires both practical skills and abstract concepts. During class time, we will cover the practical aspects and introduce the basic concepts. You will get to see the basic practical details and real examples of debugging during class sessions. Learning to debug something you've never encountered before and setting up your programming environment, for example, are *high frustration* activities, when you're learning, because you don't know what you don't know. On the other hand, diving deeper into options and more complex applications of what you have already seen in class, while challenging, is something I'm confident that you can all be successful at with minimal frustration once you've seen basic ideas in class. My goal is that you can repeat the patterns and processes we use in class outside of class to complete assignments, while acknowledging that you will definitely have to look things up and read documentation outside of class.

Each class will open with some time to review what was covered in the last session before adding new material.

To get the most out of class sessions, you should have a laptop with you. During class you should be following along with Dr. Brown. You'll answer questions on Prismia chat, and when appropriate you should try running necessary code to answer those questions. If you encounter errors, share them via Prismia chat so that we can see and help you.

After class

After class, you should practice with the concepts introduced.

This means reviewing the notes: both yours from class and the annotated notes posted to the course website.

When you review the notes, you should be adding comments on tricky aspects of the code and narrative text between code blocks in markdown cells.

While you review your notes and the annotated course notes, you should also read the documentation for new modules, libraries, or functions introduced in that class. We will collaboratively annotate notes for this course. Dr. Brown will post a basic outline of what was covered in class and we will all fill in explanations, tips, and challenge questions. Responsibility for the main annotation will rotate.

If you find anything hard to understand or unclear, write it down to bring to class the next day or post an issue on the course website.

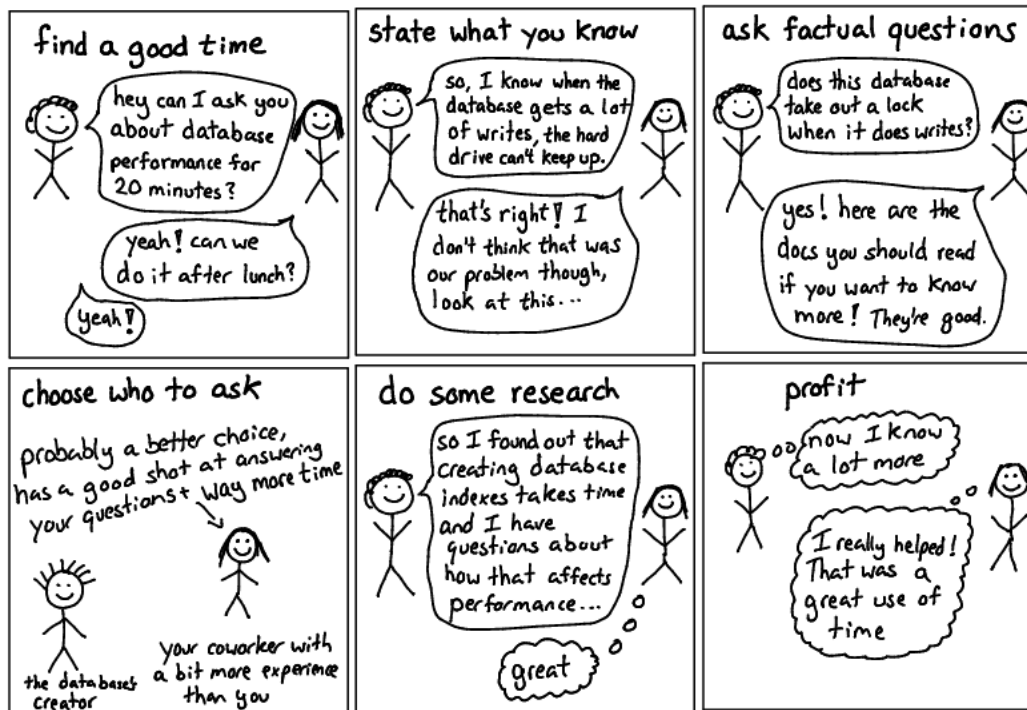
Getting Help with Programming

This class will help you get better at reading errors and understanding what they might be trying to tell you. In addition here are some more general resources.

Asking Questions

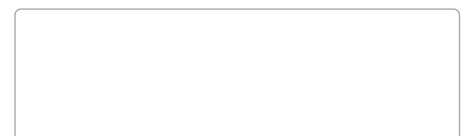
JULIA EVANS
@b0rk

asking good questions



One of my favorite resources that describes how to ask good questions is [this blog post](#) by Julia Evans, a developer who writes comics about the things she learns in the course of her work and publisher of [wizard zines](#).

Describing what you have so far



Note

A fun version of this is [rubber duck debugging](#)

Stackoverflow is a common place for programmers to post and answer questions. As such, they have written a good [guide on creating a minimal, reproducible example](#).

Creating a minimal reproducible example may even help you debug your own code, but if it does not, it will definitely make it easier for another person to understand what you have, what your goal is, and what's working.

Getting Organized for class

The only **required** things are in the Tools section of the syllabus, but this organizational structure will help keep you on top of what is going on.

Your username will be appended to the end of of the repository name for each of your assignments in class.

File structure

I recommend the following organization structure for the course:

```
CSC392
| - notes
| - kwl-char-username
| - spring2022
| - ...
```

This is one top level folder will all materials in it. A folder inside that for in class notes, and one folder per repository that you work on.

Finding repositories on github

Each assignment repository will be created on GitHub with the **introcompsys** organization as the owner, not your personal account. Since your account is not the owner, they do not show on your profile.

Your assignment repositories are all private during the semester. At the end, you may take ownership of your portfolio[^pttrans] if you would like.

If you go to the main page of the [organization](#) you can search by your username (or the first few characters of it) and see only your repositories.

Warning

Don't try to work on a repository that does not end in your username; those are the template repositories for the course and you don't have edit permission on them.