About this Site

Contents

Syllabus

Notes

Activities

FAQ

Resources

Welcome to the course website for Computer Systems and Programming Tools in Spring 2023 with Professor Brown.

This class meets TTh 12:30-1:45 in Morris Hall 323.

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

[course calendar](#)

> 💡 **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.

> **💡 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.

> **ⓘ 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

> **ⓘ 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.

Computer Systems and Programming Tools

About this course

In this course we will study the tools that we use as programmers and use them as a lens to study the computer system itself. We will begin with two fundamental tools: version control and the shell. We will focus on git and bash as popular examples of each. Sometimes understanding the tools requires understanding an aspect of the system, for example git uses cryptographic hashing which requires understanding number systems. Other times the tools helps us see how parts work: the shell is our interface to the operating system.

About this syllabus

This syllabus is a *living* document. 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.

About your instructor

Name: Dr. Sarah M Brown Office hours: listed on communication page

Dr. Sarah M Brown is a third 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

On BrightSpace, you will find links to other resource, this site and others. Any links that are for private discussion among those enrolled in the course will be available only from our course [Brightspace site](#) .

> **ⓘ Note**
>
> Seeing the BrightSpace site requires logging in with your URI SSO and being enrolled in the course

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.

> **⚠ Important**
>
> Prismia is **only** for use during class, we do not read messages there outside of class time

You can get a transcript from class from Prismia.chat using the menu in the top right.

Course Website

The course website 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.

You will also need the [gh CLI](#). It will help with authentication and allow you to work with other parts of github besides the core git operations.

Programming Environment

In this course, we will use several programming environments. In order to participate in class and 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)
- a C compiler
- [Git](#)
- A bash shell
- A web browser compatible with [Jupyter Notebooks](#)
- nano text editor (comes with GitBash and default on MacOS)
- one IDE with git support (default or via extension)
- [the GitHub CLI](#) on all OSs

Recommendation

Windows- option A   **MacOS**   **Linux**   **Chrome OS**

- Install python via [Anaconda](#) [video install](#)
- Git and Bash with [GitBash](#) ([video instructions](#)).

Zoom

(backup only & office hours only)

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 office hours and any online class sessions if needed best if you download the [Zoom client](#) on your computer. Please [log in](#) and [configure your](#)

. Please add a photo (can be yourself or something you like) 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.

Grading

This section of the syllabus describes the principles and mechanics of the grading for the course. The course is designed around your learning so the grading is based on you demonstrating how much you have learned.

Additionally, since we will be studying programming tools, we will use them to administer the course. To give you a chance to get used to the tools there will be a grade free zone for the first few weeks.

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. Apply common design patterns and abstractions to understand new code bases, tools, and components of systems.
2. 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
3. Identify the computational pipeline from hardware to high level programming language
4. Discuss implications of choices across levels of abstraction
5. 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 designed to reflect how deeply you learn the material, even if it takes you multiple attempts to truly understand a topic. The topics in this course are all topics that will come back in later courses, so it is important that you understand each of them correctly so that it helps in the next course.

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 material. You will be required to demonstrate understanding of the connections between ides from different parts of the course.

- Earning a C in this class means you have a general understanding; you will know what all the terms mean; you could follow along in a meeting where others were discussing systems concepts and use core tools for common tasks. You know where to start when looking things up.
- Earning a B means that you can 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; you can know where to start and can form good hypotheses about why uncommon errors have occurred; you can confidently figure out new complex systems.

The course is designed for you to *succeed* at a level of your choice. As you accumulate knowledge, the grading in this course is designed to be cumulative instead of based on deducting points and averaging. No matter what level of work you choose to engage in, you will be expected to revise work

until it is correct. The material in this course will all come back in other 300 and 400 level CSC courses, so it is essential that you do not leave this course with misconceptions, as they will make it harder for you to learn related material later.

Penalty-free Zone

Since learning developer tools is a core learning outcome of the course, we will also use them for all aspects of administering the course. This will help you learn these tools really well and create accountability for getting enough practice with core operations, but it also creates a high stakes situation: even submitting your work requires you understanding the tools. This would not be very fair at the beginning of the semester.

For the first three weeks we will have a low stakes penalty-free zone where we will provide extra help and reminders for how to get feedback on your work. In this period, deadlines are more flexible as well.

> ⚠️ **Important**
>
> If there are terms in the rest of this section that do not make sense while we are in the penalty-free zone, do not panic. This zone exists to help you get familiar with the terms needed.

During the third week, you will create a course plan where you establish your goals for the course and I make sure that you all understand the requirements to complete your goals.

Learning Badges

Your grade will be based on you choosing to work with the material at different levels and participating in the class community in different ways. Each of these represents different types of badges that you can earn as you accumulate evidence of your learning and engagment.

- experience: guided in class acitivies
- review: just the basics
- practice: a little bit more indepdendent
- explore: posing your own directions of inquiry
- build: in depth- application

To earn a D you must complete:

- 24 experience badges

To earn a C you must complete:

- 24 experience badges
- 18 review badges

To earn a B you must complete:

- 24 experience badges
- your choice:
    - 18 practice badges
    - 12 review + 12 practice

For an A you must complete:

- 24 experience badges
- your choice:
    - 18 practice badges + 9 explore badges
    - 18 review badges + 3 build badges
    - 6 review badges + 12 practice badges + 6 explore badges + 1 build badges
    - 12 review badges + 6 practice badges+ 3 explore badges + 2 build badges

You can also mix and match to get +/-. For example (all examples assume 24 experience badges)

- A-: 18 practice + 6 explore
- A-: 6 review + 12 practice + 9 explore
- B-: 6 review + 12 practice
- B+: 24 practice
- C+: 12 review + 6 practice

> ⚠️ **Warning**
>
> These counts assume that the semester goes as planned and that there are 26 available
> badges of each base type (experience, review, practice). If the number of available badges
> decreases by more than 2 for any reason (eg snowdays, instructor illness, etc) the threshold
> for experience badges will be decreased.

> ⚠️ **Important**
>
> There will be 20 review and practice badges available after the penalty free zone. This means
> that missing the review and practice badges in the penalty free zone cannot hurt you.
> However, it does not mean it is a good idea to not attempt them, not attempting them at all will
> make future badges harder, because reviewing early ideas are important for later ideas.

You cannot earn both practice and review badges for the same class session, but most practice badge
requirements will include the review requirements plus some extra steps. At the end of the semester,
there will be special *integrative* badge opportunities that have multipliers attached to them. These
badges will count for more than one. For example an integrative 2x review badge counts as two review
badges. These badges will be more complex than regular badges and therefore count more.

Experience Badges

You earn an experience badge in class by:

- preparing for class
- following along with the activity (creating files, using git, etc)
- responding to 80% of inclass questions (even incorrect or :idk:)
- reflecting on what you learned
- asking a question at the end of class

You can make up an experience badge by:

- preparing for class
- reading the posted notes
- completing the activity from the notes
- producing an "experience report" OR attending office hours

An experience report is evidence you have completed the activity and reflection questions. The exact
form will vary per class, if you are unsure, reach out ASAP to get instructions. These are evaluated
only for completeness/ good faith effort. Revisions will generally not be required, but clarification and
additional activity steps may be adcised if your evidence suggests you may have missed a step.

Review and Practice Badges

The tasks for these badges will be defined at the bottom of the notes for each class session *and*
aggregated to badge-type specific pages on the left hand side.

You can earn review and practice badges by:

- creating an issue for the badge you plan to work on
- completing the tasks
- submitting files to your KWL on a new branch

- creating a PR, linking the issue, and requesting a review
- revising the PR until it is approved
- merging the PR after it is approved

You should create one PR per badge

The key difference between review and practice is the depth of the activity. Work submitted for review and practice badges will be assessed for correctness and completeness. Revisions will be common for these activities, because understanding correctly, without misconceptions, is important.

> ⚠️ **Important**
>
> Revisions are to help you improve your work **and** to get used to the process of making revisions. Even excellent work can be improved. The **process** of making revisions and taking good work to excellent or excellent to exceptional is a useful learning outcome. It will help you later to be really good at working through PR revisions; we will use the same process as code reviews in industry, even though most of it will not be code alone.

Explore Badges

Explore badges require you to pose a question of your own that extends the topic. For inspiration, see the practice tasks and the questions after class.

Details and more ideas are on the [explore](#) page.

You can earn an explore badge by:

- creating an issue proposing your idea (consider this ~15 min of work or less)
- adjusting your idea until given the proceed label
- completing your exploration
- submitting it as a PR
- making any requested changes
- merging the PR after approval

For these, ideas will almost always be approved, the proposal is to make sure you have the right scope (not too big or too small). Work submitted for explore badges will be assessed for depth beyond practice badges and correctness. Revisions will be more common on the first few as you get used to them, but typically decraese as you learn what to expect.

> ⚠️ **Important**
>
> Revisions are to help you improve your work **and** to get used to the process of making revisions. Even excellent work can be improved. The **process** of making revisions and taking good work to excellent or excellent to exceptional is a useful learning outcome. It will help you later to be really good at working through PR revisions; we will use the same process as code reviews in industry, even though most of it will not be code alone.

You should create one PR per badge

Build Badges

Build badges are for when you have an idea of something you want to do. There are also some ideas on the [build](#) page.

You can earn a build badge by:

- creating an issue proposing your idea and iterating until it is given the "proceed" label
- providing updates on your progress
- completing the build
- submitting a summary report as a PR linked to your proposal issue

- making any requested changes
- merging the PR after approval

You should create one PR per badge

For builds, since they're bigger, you will propose intermediate milestones. Advice for improving your work will be provided at the milestones and revisions of the compelte build are uncommon. If you do not submit work for intermediate review, you may need to revise the complete build. The build proposal will assessed for relevance to the course and depth. The work will be assessed for completeness in comparison to the propsal and correctness. The summary report will be assessed only for completeness, revisions will only be requested for skipped or incomplete sections.

Community Badges

Community badges are awarded for extra community participation. Both programming and learning are most effective in good healthy collaboration. Since being a good member of our class community helps you learn (and helps others learn better), some collaboration is required in other badges. Some dimensions of community participation can only be done once, for example fixing a typo on the course website, so while it's valuable, all students cannot contribute to the course community in the same way. To reward these unique contributions, you can earn a community badge.

Community badges can replace missed experience, review, and practice badges, upgrade a review to a practice badge, or they can be used as an alternate way to earn a + modifier on a D,C,or B (URI doesn't award A+s, sorry). Community badges are smaller, so they are not 1:1 replacements for other badges. You can earn a maximum of 14 community badges, generally one per week. Extra helpful contributions may be awarded 2 community badges, but that does not increase your limit. When you earn them, you can plan how you will use it, but they will only be officially applied to your grade at the end of the semester. They will automatically be applied in the way that gives you the maximum benefit.

Community Badge values:

- 3 community = 1 experience badge
- 5 community = 1 review
- 7 community = 1 practice.
- 5 community badges + 1 review = 1 practice.
- 10 community = + step to a D,C, or B, **note that this is more efficient.**

You can earn community badges by:

- fixing small issues on the course website (during only)
- contributing extra terms or reviews to your team repo
- sharing articles and discussing them in the course discussions
- contributing annotated resources the course website

> ℹ️ **Note**
>
> Some participation in your group repo and a small number of discussions will be required for experience, review, and practice badges. This means that not every single contribution or peer review to your team repo will earn a community badge.

Example(nonexhaustive) uses:

- 22 experience + 17 review + 11 community = C (replace 2 experience, 1 review)
- 24 experience + 17 review + 5 community = C (replace 1 review)
- 24 experience + 18 review + 10 community = C+ (modifier)
- 24 experience + 18 practice + 10 community = B+ (modifier)
- 23 experience + 18 practice + 13 community = B+ (modifier, replace 1 experience)
- 24 experience + 16 practice + 2 review + 10 community = B (upgrade 2 review)

- 24 experience + 10 review + 10 community + 6 practice + 3 explore + 2 build = A (replace 2 review)
- 24 experience + 14 review + 10 community + 4 practice + 3 explore + 2 build = A (upgrade 2 review to practice)
- 24 experience + 12 review + 14 community + 4 practice + 3 build =A (replace 2 practice)

These show that community badges can save you work at the end of the semester by reducing the number of practice badges or simplifying badges

Deadlines

There will be fixed feedback hours each week, if your work is submitted by the start of that time it will get feedback. If not, it will go to the next feedback hours.

Experience badges

Prepare for class tasks must be done before class so that you are prepared. Missing a prepare task could require you to do an experience report to make up what you were not able to do in class.

If you miss class, the experience report should be at least attempted/drafted (though you may not get feedback/confirmation) before the next class that you attend. This is strict, not as punishment, but to ensure that you are able to participate in the next class that you attend. Skipping the experience report for a missed class, may result in needing to do an experience report for the next class you attend to make up what you were not able to complete due to the missing class activities.

If you miss multiple classes, create a catch-up plan to get back on track by contacting Dr. Brown.

Review and Practice Badges

These badges have 5 stages:

- posted: tasks are on the course website
- planned: an issue is created
- started: one task is attempted and a draft PR is open
- completed: all tasks are attempted PR is ready for review, and a review is requested
- earned: PR is approved (by instructor or a TA) and work is merged

> 💡 **Tip**
>
> these badges *should* be reviewed and started before the next class. This will set you up to make the most out of each class session. However, only prepare for class tasks have to be done immediately.

These badges badges must be *started* within one week of when the are posted and *completed* within two weeks. A task is attempted when you have answered the questions or submitted evidence of doing an activity or asked a sincere clarifying question.

If a badge is planned, but not started within one week it will become expired and ineligble to be earned. You may request extensions to complete a badge by updating the PR message, these will typically be granted. Extensions for starting badges will only be granted in exceptional circumstances.

Once you have a good-faith attempt at a complete badge, you have until the end of the semester to finish the revisions in order to *earn* the badge.

> 💡 **Tip**
>
> Try to complete revisions quickly, it will be easier for you

Explore Badges

Explore badges have stages:

- proposed: issue created
- in progress: issue is labeled "proceed" by the instructor
- complete: work is complete, PR created, review requested
- revision: "request changes" review was given
- earned: PR approved

Explore badges are feedback-limited. You will not get feedback on subsequent explore badge proposals until you earn the first one. Once you have one earned, then you can have up to two in progress and two in revision at any given time.

Build Badges

You may earn at most one build badge per month, with final grading in May. To earn three build badges, you must earn the first one by the end of March.

Schedule

Overview

The following is a tentative 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.

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 getting started with the programming tools.

What tools do Computer Scientists use?

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. While studying the tools and how they work, we will get to see how some common abstractions are re-used throughout the fields and it gives a window and good motivation to begin considering how the computer actually works.

Topics:

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

What Happens When I run code?

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

- compiliation
- linking
- basic hardware components

Tentative Schedule

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 overview. More Details will be in other parts of the site, linked to here.

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, FIXME. 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 FIXME, 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,](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 you are able to successfully complete the course.

Office Hours & Communication

Help Hours

```
/tmp/ipykernel_1904/2146052215.py:1: FutureWarning: this method is deprecated in
favour of `Styler.hide(axis="index")`
  help_df.style.hide_index()
```

day	time	location	host
TBA	TBA	TBA	Dr. Brown
TBA	TBA	TBA	Mark
TBA	TBA	TBA	Marcin

Online office hours locations are linked on the [GitHub Organization Page](#)

> ⚠️ **Important**
>
> You can only see them if you are a "member" you will be invited in the first week of the
> course. If you do not accept via the e-mail invite, you can accept your invite from that page for
> 7 days. After that, ask for a new invite by e-mailing `cscsystools-help-group@uri.edu`

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 a 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. Welcome and Introduction

1.1. Introductions

You can see more about me in the about section of the syllabus.

I look forward to getting to know you all better.

1.2. Prismia

- instead of slides
- you can message us
- we can see all of your responses

- emoji!

Are emoji fun or do they make me Old? *no penalty, this is for fun & to practice*

- [ ] fun
- [ ] not cool

questions can also be "graded"

- this is instant feedback
- participation will be checked, not impact your final grade
- this helps both me and you know how you are doing

What is the topic of this course?

- [ ] hardware
- [ ] programming
- [x] computer systems and programming tools

or open ended

What is one thing you want the TAs and I to know about you?

1.2.1. Some Background

- What programming environments do you have?
- What programming environments are you most comfortable with?

what programming tools are you familiar with? **send exactly one tool name per message, but you can send multiple messages**

This information will help me prepare

1.2.2. My focus is for you to learn

- that means, practice, feedback, and reflection
- you should know that you have learned
- you should be able to apply this material in other courses

1.3. Getting started with KWL charts

Your [KWL](#) chart is where you will start by tracking what you know now/before we start and what you want to learn about each topic. Then you will update it throughout the semester.

Today we did the following:

1. Accept the assignment to create your repo: [KWL Chart](#)
2. Edit the README (only file there) to add your name by clicking the pencil icon ([editing a file](#) step 2)
3. adding a descriptive commit message ([editing a file](#) step 5)
4. created a new branch (named `priorknowledge`) ([editing a file](#) step 7-8)
5. added a message to the Pull Request ([pull request](#) step 5)
6. Creating a pull request ([pull request](#) step 6)
7. Clicking Merge Pull Request

> **ⓘ Further Reading**
>
> GitHub Docs are really helpful and have screenshots
>
> - [editing a file](#)
> - [pull request](#)

1.4. Git and GitHub terminology

We also discussed some of the terminology for git. We will also come back to these ideas in greater detail later.

1.5. What is this course about?

In your KWL chart, there are a lot of different topics that are not obviously related, so what is this course really about?

- practical exposure to important tools
- design features of those tool categories
- basic knowledge of many parts of the CS core
- focus on the connections

We will use learning the tools to understand how computer scientists think and work.

Then we will use the tools to examine the field of Computer Science top to bottom (possibly out of order).

1.5.1. How it fits into your CS degree

In CSC110, you learn to program in python and see algorithms from a variety of domain areas where computer science is applied.

Then in CSC 340 and 440 you study the algorithms more mathematically, their complexity, etc.

In CSC211, 212, you learn the foundations of computer science: general programming and data structures.

Then in 301, 305, 411, 412 you study different aspects of software design and how computers work.

In this class, we're going to connect different ideas. We are going to learn the tools used by computer scientists, deeply. You will understand why the tools are the way they are and how to use them even when things go wrong.

> 💡 **Tip**
>
> knowing where you've been and where we're going will help you understand and remember

1.6. Course Admin

1.6.1. Programming is Collaborative

There are two very common types of collaboration

- code review (workign independently and then reviewing)
- pair programming (sitting together and discussing while writing)

We are going to build your skill in the *code review* model. This means you need to collaborate, but collaboration in school tends to be more stressful than it needs to. If students have different goals or motivation levels it can create conflict. So **you will have no group graded work** but you will get the chance to work on something together in a low stakes way.

You will have a "home team" that you work with throughout the semester to build a glossary and a "cookbook" of systems recipes.

Your contributions and your **peer reviews** will be assessed individually for your grade, but you need a team to be able to practice these collaborative aspects.

[team formation survey](#)

> ⚠️ **Important**
>
> Remember to fill out the [team formation survey](#)

1.6.2. Class forum

This [community repository "assignment"](#) will add you to a "team" with the whole class. It allows us to share things on GitHub, for the whole class, but not the whole internet.

1.6.3. Get Credit for Today's class

1. Run your Experience Reflection (incalss) action on your kwl repo
2. today's evidence is your KWL repo existing and having the commits as above

1.7. Review today's class

1. Review the notes after I post them.
2. Fill in the first two columns of your KWL chart.
3. [review git and github vocabulary](#) (include link in your badge PR)

1.8. Prepare for Next Class

1. Read the syllabus section of the course website carefully and explore the whole course [website](#)
2. Bring questions about the course to class
3. Think about one thing you've learned really well (computing or not). Be prepared to discuss the following: How do you know that you know it? What was it llike to first learn it?
4. Post an introduction to your classmates [on our discussion forum](#)

1.9. More Practice

1. Review the notes after I post them.
2. Fill in the first two columns of your KWL chart.
3. [review git and github vocabulary](#) (include link in your badge PR)
4. Read more about [version control in general](#) and add a "version control" row to your KWL chart with all 3 columns filled in.

1.10. Questions After Today's Class

> ℹ️ **Note**
>
> I will add the rest later, today I had a one time conflict.

1.10.1. How to directly merge all suggestions without clicking commit suggestion?

Unfortunately, that is not an option on a PR review, but in general, you will not make a lot of changes in a review. We will learn other ways to do this

1.10.2. What is the importance of github in the real world? Is it so people can collaborate on code together, or maybe its somewhere to share your code and help others/inspire

other code writers or a combination of both?

1.10.3. How do I add collaborators to my repository?

1.10.4. How does the grading work in this class?

KWL Chart

Working with your KWL Repo

1. Work on a specific branch for each activity you work on
2. when it is ready for review, create a PR from the item-specifc branch to `main`.
3. when it is approved, merge into main.

Minimum Rows

```
# KWL Chart


<!-- replace the  _ in the table or add new rows as needed -->

| Topic | Know | Want to Know | Learned |
| ------| ------- | ------ | ------- |
| Git | _ | _ | _ |
| GitHub | _ | _ | _ |
| Terminal | _ | _ | _ |
| IDE | _ | _ | _ |
| text editors | _ | _ | _ |
|file system | _ | _ |_ |
|bash | _ | _ | _ |
|abstraction | _ | _ | _ |
|programming languages | _ | _ | _ |
|git workflows | _ | _ | _ |
| git branches | _ | _ | _ |
| bash redirects | _ | _ | _ |
|number systems | _ | _ | _ |
| merge conflicts | _ | _ | _ |
| documentation | _ | _ | _ |
| templating | _ | _ | _ |
|bash scripting | _ | _ | _ |
| developer tools | _ | _ | _ |
| networking | _ | _ | _ |
|ssh | _ | _ | _ |
| ssh keys | _ | _ | _ |
|compiling | _ | _ | _ |
| linking   | _ | _ | _ |
| building | _ | _ | _ |
| machine representation  | _ | _ | _ |
| integers   | _ | _ | _ |
| floating point  | _ | _ | _ |
|logic gates | _ | _ | _ |
| ALU | _ | _ | _ |
| binary operations | _ | _ | _ |
| memory | _ | _ | _ |
| cache | _ | _ | _ |
| register | _ | _ | _ |
| clock | _ | _ | _ |
| Concurrency | _ | _ | _ |
```

Required Files

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[1], line 3
      1 import pandas as pd
----> 3 check_df = pd.read_csv('kwl.csv')
      5 # FIXME: update grade free dates
      6 penalty_free_dates = ['2022-01-24','2023-01-26']

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-
packages/pandas/util/_decorators.py:211, in deprecate_kwarg.
<locals>._deprecate_kwarg.<locals>.wrapper(*args, **kwargs)
    209     else:
    210             kwargs[new_arg_name] = new_arg_value
--> 211 return func(*args, **kwargs)

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-
packages/pandas/util/_decorators.py:331, in deprecate_nonkeyword_arguments.
<locals>.decorate.<locals>.wrapper(*args, **kwargs)
    325 if len(args) > num_allow_args:
    326     warnings.warn(
    327         msg.format(arguments=_format_argument_list(allow_args)),
    328         FutureWarning,
    329         stacklevel=find_stack_level(),
    330     )
--> 331 return func(*args, **kwargs)

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-
packages/pandas/io/parsers/readers.py:950, in read_csv(filepath_or_buffer, sep,
delimiter, header, names, index_col, usecols, squeeze, prefix, mangle_dupe_cols,
dtype, engine, converters, true_values, false_values, skipinitialspace, skiprows,
skipfooter, nrows, na_values, keep_default_na, na_filter, verbose,
skip_blank_lines, parse_dates, infer_datetime_format, keep_date_col, date_parser,
dayfirst, cache_dates, iterator, chunksize, compression, thousands, decimal,
lineterminator, quotechar, quoting, doublequote, escapechar, comment, encoding,
encoding_errors, dialect, error_bad_lines, warn_bad_lines, on_bad_lines,
delim_whitespace, low_memory, memory_map, float_precision, storage_options)
    935 kwds_defaults = _refine_defaults_read(
    936     dialect,
    937     delimiter,
    (...)
    946     defaults={"delimiter": ","},
    947 )
    948 kwds.update(kwds_defaults)
--> 950 return _read(filepath_or_buffer, kwds)

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-
packages/pandas/io/parsers/readers.py:605, in _read(filepath_or_buffer, kwds)
    602 _validate_names(kwds.get("names", None))
    604 # Create the parser.
--> 605 parser = TextFileReader(filepath_or_buffer, **kwds)
    607 if chunksize or iterator:
    608     return parser

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-
packages/pandas/io/parsers/readers.py:1442, in TextFileReader.__init__(self, f,
engine, **kwds)
   1439     self.options["has_index_names"] = kwds["has_index_names"]
   1441 self.handles: IOHandles | None = None
-> 1442 self._engine = self._make_engine(f, self.engine)

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-
packages/pandas/io/parsers/readers.py:1735, in TextFileReader._make_engine(self,
f, engine)
   1733     if "b" not in mode:
   1734         mode += "b"
-> 1735 self.handles = get_handle(
   1736     f,
   1737     mode,
   1738     encoding=self.options.get("encoding", None),
   1739     compression=self.options.get("compression", None),
   1740     memory_map=self.options.get("memory_map", False),
   1741     is_text=is_text,
   1742     errors=self.options.get("encoding_errors", "strict"),
   1743     storage_options=self.options.get("storage_options", None),
   1744 )
   1745 assert self.handles is not None
   1746 f = self.handles.handle

File /opt/hostedtoolcache/Python/3.8.16/x64/lib/python3.8/site-
packages/pandas/io/common.py:856, in get_handle(path_or_buf, mode, encoding,
compression, memory_map, is_text, errors, storage_options)
    851 elif isinstance(handle, str):
    852     # Check whether the filename is to be opened in binary mode.
    853     # Binary mode does not support 'encoding' and 'newline'.
    854     if ioargs.encoding and "b" not in ioargs.mode:
    855         # Encoding
--> 856         handle = open(
```

```
857                handle,
858                ioargs.mode,
859                encoding=ioargs.encoding,
860                errors=errors,
861                newline="",
862            )
863        else:
864            # Binary mode
865            handle = open(handle, ioargs.mode)

FileNotFoundError: [Errno 2] No such file or directory: 'kwl.csv'
```

Team Repo

Contributions

Your team repo is a place to build up a glossary of key terms and a "cookbook" of "recipes" of common things you might want to do on the shell, bash commands, git commands and others.

For the glossary, follow the jupyterbook syntax.

For the cookbook, use standard markdown.

to denote code inline `use single backticks`

```
to denote code inline `use single backticks`
```

to make a code block use 3 back ticks

```
```
to make a code block use 3 back ticks
```
```

To nest blocks use increasing numbers of back ticks.

To make a link, `[show the text in squarebrackets](url/in/parenthesis)`

Collaboration

You will be in a "team" that is your built in collaboration group to practice using Git Collaboratively. There will be assignments that are to be completed in that repo as well. These activities will be marked accordingly. You will take turns and each of you is required to do the initialization step on a recurring basis.

This is also where you can ask questions and draft definitions to things.

Peer Review

If there are minor errors/typos, suggest corrections inline.

In your summary comments answer the following:

- Is the contribution clear and concise? Identify any aspect of the writing that tripped you up as a reader.
- Are the statements in the contribution verifiable (either testable or cited source)? If so, how do you know they are correct?
- Does the contribution offer complete information? That is, does it rely on specific outside knowledge or could another CS student not taking our class understand it?
- Identify one strength in the contribution, and identify one aspect that could be strengthened further.

Choose an action:

- If the suggestions necessary before merging, select **request changes**.

- If it is good enough to merge, mark it **approved** and open a new issue for the broader suggestions.
- If you are unsure, post as a **comment** and invite other group members to join the discussion.

Review Badges

Review After Class

After each class, you will need to review the day's material. This includes reviewing prismia chat to see any questions you got wrong and reading the notes. Most days there will be specific additional activities and questions to answer. These should be in your KWL repo. Review activities will help you to reinforce what we do in class and guide you to practice with the most essential skills of this class.

2023-01-24

[related notes](#)

Activities:

1. Review the notes after I post them.
2. Fill in the first two columns of your KWL chart.
3. [review git and github vocabulary](#) (include link in your badge PR)

Prepare for the next class

These tasks are not always based on things that we have already done. Sometimes they are to have you start thinking about the topic that we are *about* to cover. Getting whatever you know about the topic fresh in your mind in advance of class will help what we do in class stick for you when we start.

The correct answer is not as important for these activities as it is to do them before class. We will build on these in class.

2023-01-24

[related notes](#)

Activities:

More Practice Badges

> ℹ️ **Note**
>
> these are listed by the date they were *posted*

More practice exercises are a chance to try new dimensions of the concepts that we cover in class.

> ℹ️ **Note**
>
> Activities will appear here once the semester begins

2023-01-24

[related notes](#)

Activities:

1. Review the notes after I post them.
2. Fill in the first two columns of your KWL chart.
3. [review git and github vocabulary](#) (include link in your badge PR)
4. Read more about [version control in general](#) and add a "version control" row to your KWL chart with all 3 columns filled in.

Explore Badges

> ⚠️ **Warning**
>
> Explore Badges are not required, but an option for higher grades. The logistics of this could be streamlined or the instructions may become more detialed during the penalty free zone.

Explore Badges can take different forms so the sections below outline some options. This page is not a cumulative list of requirements or an exhaustive list of options.

> 💡 **Tip**
>
> You might get a lot of suggestions for improvement on your first one, but if you apply that advice to future ones, they will get approved faster.

How do I propose?

If you propose something too big, you might be advised to consider a build badge instead. If you propose something too small, you will get ideas as options for how to expand it and you pick which ones.

Where to put the work?

- If you extend a more practice exercise, you can add to the markdown file that the exercise instructs you to create.
- If its a question of your own, add a new file to your KWL repo.

> ❗ **Important**
>
> Either way, there must be a separate issue for this work that is also linked to your PR

What should the work look like?

It should look like a blog post, written tutorial, graphic novel, or visual aid with caption. It will likely contain some code excerpts the way the class notes do. Style-wise it can be casual, like how you may talk through a concept with a friend or a more formal, academic tone. What is important is that it clearly demonstrates that you understand the material.

The exact length can vary, but these must go beyond what we do in class in scope

Explore Badge Ideas:

- Extend a more practice:
  - for a more practice that asks you to describe potential uses for a tool, try it out, find or write code excerpts and examine them
  - for a more practice that asks you to try something, try some other options and compare and contrast them. eg "try git in your favorite IDE" -> "try git in three different IDEs, compare and contrast, and make recommendations for novice developers"

- For a topic that left you still a little confused or their was one part that you wanted to know more about. Details your journey from confusion or shallow understanding to a full understanding. This file would include the sources that you used to gather a deeper understanding. eg:
  - Describe how cryptography evolved and what caused it to evolve (i.e. SHA-1 being decrypted)
  - Learn a lot more about a specific number system
  - compare another git host
  - try a different type of version control
- Create a visual aid/memory aid to help remember a topic. Draw inspriation from Wizard Zines or
- Review a reference or resource for a topic

Examples from past students:

- Scripts/story boards for tiktoks that break down course topics
- Visual aid drawings to help remember key facts

For special formatting, use [jupyter book's documentation](#).

ℹ Note

These students technically submitted these under different grading structures, but were approximately the same as the explore badges

Syllabus and Grading FAQ

How much does activity x weigh in my grade?

There is no specific weight for any activities, because your grade is based on earning the badges. Everything at a level must be complete and correct.

How do I keep track of my earned badges?

You will have several options. You will have a project board that you can track assigned work, in progress work and earned badges with in one place. This is quite different than checking your grade in BrightSpace, but using tools like this represents the real tools used by developers.

Additionally, when we log them in our private gradebook, we will give you a "receipt" that is 128 characters long. You will be able to use provided command line tools and github actions to produce a report of your status at any time from those receipts. Additionally, at particular points in the course, an in class or class preparation activity will be for you to review a "progress report" that we give you and update your success plan for the course.

I don't understand the feedback on this assignment

If you have questions about your feedback, the best thing to do is to start a discussion on the PR for that work. Either reply directly to one of the inline comments, or the summary.

Be specific about what you think is wrong so that we can expand more.

What should a Deeper exploration look like and where do I put it?

It should be a tutorial or blog style piece of writing, likely with code excerpts or screenshots embedded in it.

[an example that uses mostly screenshots](#)

[an example of heavily annotated code](#)

They should be markdown files in your KWL repo. I recommend myst markdown.

Git and GitHub

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

If your error looks like this…

```
! [rejected] main -> main (fetch first)
error: failed to push some refs to <repository name>
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Your local version and github version are out of sync, you need to pull the changes from github to your local computer before you can push new changes there.

After you run

```
git pull
```

You'll probably have to resolve a merge conflict

My command line says I cannot use a password

GitHub has strong rules about authentication You need to use SSH with a public/private key; HTTPS with a Personal Access Token or use the GitHub CLI auth

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

That's ok. You can fix it.

You'll have to work offline and use GitHub in your browser together for this fix. The following instuctions will work in terminal on Mac or Linux or in GitBash for Windows. (see Programming Environment section on the tools page).

First get the url to clone your repository (unless you already have it cloned then skip ahead): on the main page for your repository, click the green "Code" button, then copy the url that's show
screenshot of copying .git url

Next open a terminal or GitBash and type the following.

```
git clone
```

then past your url that you copied. It will look something like this, but the last part will be the current assignment repo and your username.

```
git clone https://github.com/rhodyprog4ds/portfolio-brownsarahm.git
```

When you merged the Feedback pull request you advanced the `feedback` branch, so we need to hard reset it back to before you did any work. To do this, first check it out, by navigating into the folder for your repository (created when you cloned above) and then checking it out, and making sure it's up to date with the `remote` (the copy on GitHub)

```
cd portfolio-brownsarahm
git checkout feedback
git pull
```

Now, you have to figure out what commit to revert to, so go back to GitHub in your browser, and swithc to the feedback branch there. Click on where it says `main` on the top right next to the branch icon and choose feedback from the list.

screenshot of switching to the feedback branch on github

Now view the list of all of the commits to this branch, by clicking on the clock icon with a number of commits

screenshot of clicking to view commits

On the commits page scroll down and find the commit titled "Setting up GitHub Classroom Feedback" and copy its hash, by clicking on the clipboard icon next to the short version.

screenshot copying hash

Now, back on your terminal, type the following

```
git reset --hard
```

then paste the commit hash you copied, it will look something like the following, but your hash will be different.

```
git reset --hard 822cfe51a70d356d448bcaede5b15282838a5028
```

If it works, your terminal will say something like

```
HEAD is now at 822cfe5 Setting up GitHub Classroom Feedback
```

but the number on yours will be different.

Now your local copy of the `feedback` branch is reverted back as if you had not merged the pull request and what's left to do is to push those changes to GitHub. By default, GitHub won't let you push changes unless you have all of the changes that have been made on their side, so we have to tell Git to force GitHub to do this.

Since we're about to do something with forcing, we should first check that we're doing the right thing.

```
git status
```

and it should show something like

```
On branch feedback
Your branch is behind 'origin/feedback' by 12 commits, and can be fast-forwarded.
  (use "git pull" to update your local branch)
```

Your number of commits will probably be different but the important things to see here is that it says `On branch feedback` so that you know you're not deleting the `main` copy of your work and `Your branch is behind origin/feedback` to know that reverting worked.

Now to make GitHub match your reverted local copy.

```
git push origin -f
```

and you'll get something like this to know that it worked

```
Total 0 (delta 0), reused 0 (delta 0)
To https://github.com/rhodyprog4ds/portfolio-brownsarahm.git
 + f301d90...822cfe5 feedback -> feedback (forced update)
```

Again, the numbers will be different and it will be your url, not mine.

Now back on GitHub, in your browser, click on the code tab. It should look something like this now. Notice that it says, "This branch is 11 commits behind main" your number will be different but it should be 1 less than the number you had when you checked `git status`. This is because we reverted the

changes you made to main (11 for me) and the 1 commit for merging main into feedback. Also the last commit (at the top, should say "Setting up GitHub Classroom Feedback").

screenshot of feedback branch code tab after revert

Now, you need to recreate your Pull Request, click where it says pull request.

screenshot of pull request link highlighted

It will say there isn't anything to compare, but this is because it's trying to use `feedback` to update `main`. We want to use `main` to update `feedback` for this PR. So we have to swap them. Change base from `main` to `feedback` by clicking on it and choosing `feedback` from the list.

screenshot of changing pr base to feedback

Then the change the compare `feedback` on the right to `main`. Once you do that the page will change to the "Open a Pull Request" interface.

screenshot of making a pr

Make the title "Feedback" put a note in the body and then click the green "Create Pull Request" button.

Now you're done!

If you have trouble, create an issue and tag `@@rhodyprog4ds/fall20instructors` for help.

For an Assignment, should we make a new branch for every assignment or do everything in one branch?

```
Doing each new assignment in its own branch is best practice. In a typical software
development flow once the codebase is stable a new branch would be created for each
new feature or patch. This analogy should help you build intuition for this GitHub
flow and using branches.  Also, pull requests are the best way for us to give you
feedback. Also, if you create a branch when you do not need it, you can easily merge
them after you are done, but it is hard to isolate things onto a branch if it's on
main already.
```

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.

Begin a great programmer does not require memorizing all of the specific commands, but instead knowing the common patterns and how to use them to interpret others' code and write your own. Being efficient requires knowing how to use tools and how to let the computer do tedious tasks for you. This is how this course is designed to help you, but you have to get practice with these things.

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. These tools can help you when you are writing cod eand forget a specific bit of syntax, but these tools will not help you *read* code or debug environment issues. You also have to know how to effectively use these tools.

Knowing the common abstractions we use in computing and recognizing them when they look a little

bit differently will help you with these more complex tasks. Understanding what is common when you move from one environment to another or to 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.

Why this way?

Learning requires iterative practice. In this class, you will first get ready to learn by preparing for class. Then, in class, you will get a first experience with the material. The goal is that each class is a chance to learn by engaging with the ideas, it is to be a guided inquiry. Some classes will have a bit more lecture and others will be all hands on with explanation, but the goal is that you *experience* the topics in a way that helps you remember, because being immersed in an activity helps brains remember more than passively watching something. Then you have to practice with the material

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.

Preparing for class will be activities that helpy ou bring your prior knowledge to class in the most helpful way, help me mee

You will be making a lot of documentation of bits, in your own words. You will be directed to try things and make notes. This based on a recommended practices from working devs to [keep a notebook]] (https://blog.nelhage.com/2010/05/software-and-lab-notebooks/) or keep a blog and notebook.

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.

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

comic on asking questions, that summarizes blog post

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

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.

> **ℹ Note**
>
> A fun version of this is [rubber duck debugging](#)

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:

```
CSC310
  |- notes
  |- portfolio-username
  |- 02-accessing-data-username
  |- ...
```

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

Please **do not** include all of your notes or your other assignments all inside your portflio, it will make it harder to grade.

Finding repositories on github

Each assignment repository will be created on GitHub with the `rhodyprog4ds` organization as the owner, not your personal acount. 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.

More info on cpus

Resource	Level	Type	Summary
What is a CPU, and What Does It Do?	1	Article	Easy to read article that explains CPUs and their use. Also touches on "buses" and GPUs.
Processors Explained for Beginners	1	Video	Video that explains what CPUs are and how they work and are assembled.
The Central Processing Unit	1	Video	Video by Crash Course that explains what the Central Processing Unit (CPU) is and how it works.