About this Site

Contents

Syllabus

- Computer Systems and Programming Tools
- Tools and Resources
- Grading
- Badge Deadlines and Procedures
- Detailed Grade Calculations
- Schedule
- Support
- General Policies
- Office Hours & Communication

Activities

- KWL Chart
- Team Repo
- Review Badges
- Prepare for the next class
- More Practice Badges
- KWL File List
- Explore Badges
- Build Badges

FAQ

- Syllabus and Grading FAQ
- Git and GitHub
- Other Course Software/tools

Resources

- Glossary
- General Tips and Resources
- How to Study in this class
- GitHub Interface reference
- Language/Shell Specific References
- Getting Help with Programming
- Getting Organized for class
- More info on cpus

Skip to main content

- Advice from Spring 2022 Students

Welcome to the course website for Computer Systems and Programming Tools in Spring 2024 with Professor Brown.

This class meets TuTh 12:30-1:45 in Ranger 302 and lab on Monday 3-4:45 in Ranger 202.

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

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

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.

```
# this is a comment in a clode block
command argument --option -a
```

```
command output
important line, emphasized
```

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.

Skip to main content

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.

Click here!

Special tips will be formatted like this

Check your Comprehension

Questions to use to check your comprehension will looklike this

Contribute

Chances to earn community badges will sometimes be marked like this

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.

Should you download the syllabus and rely on your offline copy?

No, because the syallabus changes

About your instructor

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

Skip to main content

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

Land Acknowledgement

> **Important**
>
> The University of Rhode Island land acknowledgment is a statement written by members of the University community in close partnership with members of the Narragansett Tribe. For more information see the university land acknowledgement page

The University of Rhode Island occupies the traditional stomping ground of the Narragansett Nation and the Niantic People. We honor and respect the enduring and continuing relationship between the Indigenous people and this land by teaching and learning more about their history and present-day communities, and by becoming stewards of the land we, too, inhabit.

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 Brightspace.

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**

> **Not**
>
> Seein
> loggi
> being

Skip to main content

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.

> **Important**
>
> You need to install this on Mac

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)

Skip to main content

Recommendation

Windows- option A | **Windows - option B** | **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 account. 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, programming tools, and components of systems.
2. Apply appropriate programming workflows using context-relevant tools that enable adherance to best practices for effective code, developer time efficiency, and collaboration.
3. 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
4. Identify how information flows across levels of abstraction.
5. Discuss implications of design choices across levels of abstraction

Skip to main content

7. Differentiate between social conventions and technical requirements in programming contexts.

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 in the Computer Science major, 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 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.
- 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 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.

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.

If you made an error in an assignment what do you need to do?

Read the suggestions and revise the work until it is correct.

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. If work is submitted incorrectly, we will still see it because we will manually go look for all activities. After this zone, we will assume you *chose* to skip something if we do not see it submitted.

What happens if you merged a PR without feedback?

During the Penalty-Free zone, we will help you figure that out and fix it so you get credit for it. After that, you have to fix it on your own (or in office hours) in order to get credit.

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.

What happens if you're confused by the grading scheme right now?

Nothing to worry about, we will review it again in week three after you get a chance to build the right habits and learn vocabulary. There will also be a lab activity that helps us to be sure that you understand it at that time.

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 form of engagment is tracked with a particular type of badge that you can earn as you accumulate evidence of your learning and engagment.

- experience: guided in class activities, with reflection
- lab: accountability to basics through 1:1 conversation with a member of the instructional team
- review: just the basics
- practice: a little bit more indepdendent
- explore: posing your own directions of inquiry
- build: in depth application of course topics

Below are the recommended paths to different grade targets.

To earn a D, complete:

- 22 experience badges
- 13 lab check outs

To earn a C, complete:

- 22 experience badges
- 13 lab check outs
- 18 review badges

To earn a B, complete:

Skip to main content

- 13 lab check outs
- one of the following:
  - 18 practice badges
  - 12 review + 12 practice

For an A you must complete:

- 22 experience badges
- 13 lab check outs
- one of the following:
  - 18 practice badges + 6 explore badges
  - 18 review badges + 3 build badges
  - 6 review badges + 12 practice badges + 4 explore badges + 1 build badges
  - 12 review badges + 6 practice badges+ 2 explore badges + 2 build badges

You can also mix and match to get +/-. For example (all examples below assume 22+ experience badges aand 13 lab checkouts)

- A-: 18 practice + 4 explore
- B+: 6 review + 12 practice + 4 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.

All of these badges will be tracked through PRs in your kwl repo. Each PR must have a title that includes the badge type and associated date. We will use scripts over these to track your progress.

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.

In the second half 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.

Can you do any combination of badges?

Skip to main content

Experience Badges

In class

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

Makeup

You can make up an experience badge by:

- preparing for class
- reading the posted notes
- completing the activity from the notes
- completeing an “experience report”
- attaching evidence as indiated in notes OR attending office hours to show the evidence

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 advised if your evidence suggests you may have missed a step.

Do you earn badges for prepare for class?

No, prepare for class tasks are folded into your experience badges.

What do you do when you miss class?

Read the notes, follow along, and produce and experience report or attend office hours.

What if I have no questions?

Learning to ask questions is important. Your questions can be clarifying (eg because you misunderstood something) or show that you understand what we covered well enough to think of hypothetical scenarios or options or what might come next. Basically, focused curiosity.

Lab Checkouts

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 fo the course website.

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

Where do you find assignments?

At the end of notes and on the separate pages in the activities section on the left hand side

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)

Skip to main content

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.

You can see some ideas as they arise by issues labeled `community`.

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.

Skip to main content

Community Badge values:

- 3 community = 1 experience badge
- 4 community = 1 review
- 7 community = 1 practice.
- 3 community badges + 1 review = 1 practice.
- 10 community = add a `+` to a D,C, or B, **note that this is more impactful**

You can earn community badges by:

- fixing small issues on the course website (during penalty free zone 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

You will maintain a list of your contributions in your KWL repo in the `community_contributions.md` file. Every individual change to this file (representing one contribution) should be commited to a new branch and then submitted as a PR, with a review requested from @brownsarahm.

> **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

🎁 Free corrections

All work must be correct and complete to earn credit. In general, this means that when your work is not correct, we will give you guiding questions and advice so that you can revise the work to be correct. Most of the time asking you questions is the best way to help you learn, but sometimes, especially for small things, showing you a correct example is the best way to help you learn.

Additionally, on rare occasions, a student can submit work that is incorrect or will have down-the-line consquences but does not

within the `< >`. Later, we will do things within the kwl repo that will rely on the title line being filled in, but it's not a big revision where the student needs to rethink about what they submitted.

In these special occasions, good effort that is not technically correct may be rewarded with a 🎁. In this case, the instructor or TA will give a suggestion, with the 🎁 emoji in the comment and leave a review as "comment" instead of "changes requested" or "approved". If the student commits the suggestion to acknowledge that they read it, the instructor will then leave an approving review. Free corrections are only available when revisions are otherwise eligible. This means that they cannot extend a deadline and they are not available on the final grading that occurs after our scheduled "exam time".

Important

These free corrections are used at the instructional team's discretion and are not guaranteed.

This means that, for example, the same mistake the first time, might get a 🎁, a second will probably be a hint, and a third or fourth time might be a regular revision where we ask you to go review prior assignments to figure out what you need to fix with a broad hint instead of the specific suggestion

IDEA

If the course response rate on the IDEA survey is about 75%, 🎁 will be applicable to final grading. **this includes the requirement of the student to reply**

Ungrading Option

At the end of the semester, you have the option of submitting a final reflection that states what grade you think you deserve, and justifies it by summarizing what you have learned and providing evidence of that. Instructions for this option will be provided as we approach the end of the semester. The policy of no submitted content that was not generated by you still applies. If you take this option, you may be required to also take an oral exam by appointment to supplement the evidence provided in your reflection.

This option exists in recognition of the fact that grading schemes are not perfect and I am truly committed to your learning. If you think that the grading scheme described on this page is working out to you earning a different grade than you deserve and you can support that with strong evidence that you have learned, you can have the grade you deserve.

What do you think?

share your thoughts on this option in the discussions for the class and then

Badge Deadlines and Procedures

This page includes more visual versions of the information on the badge page. You should read both, but this one is often more helpful, because some of the processes take a lot of words to explain and make more sense with a diagram for a lot of people.

Skip to main content

```
%matplotlib inline
import os
from datetime import date,timedelta
import calendar
import pandas as pd
import numpy as np
import seaborn as sns
from myst_nb import glue
# style note: when I wrote this code, it was not all one cell. I merged the cells
# for display on the course website, since Python is not the main outcome of this course

# semester settings
first_day = date(2024,1,22)
last_day = date(2024,4,29)


# no_class_ranges = [(date(2023,11,23),date(2023,11,26)),
#                    (date(2023,11,13)),
#                   (date(2023,10,10))]
no_class_ranges = [(date(2024,3,10),date(2024,3,16)),
                   (date(2024,2,19))]


meeting_days =[1,3] # datetime has 0=Monday

penalty_free_end = date(2024, 2, 9)


def day_off(cur_date,skip_range_list):
    '''
    is the current date a day off?

    Parameters
    ----------
    cur_date : datetime.date
        date to check
    skip_range_list : list of datetime.date objects or 2-tuples of datetime.date
        dates where there is no class, either single dates or ranges specified by a tuple

    Returns
    -------
    day_is_off : bool
        True if the day is off, False if the day has class
    '''
    # default to not a day off
    day_is_off=False
    #
    for skip_range in skip_range_list:
        if type(skip_range) == tuple:
            # if any of the conditions are true that increments and it will never go down, flase=0, true=1
            day_is_off +=  skip_range[0]<=cur_date<=skip_range[1]
        else:
            day_is_off += skip_range == cur_date
    #
    return day_is_off


# enumerate weeks

mtg_delta = timedelta(meeting_days[1]-meeting_days[0])
week_delta = timedelta(7)



possible = [(first_day+week_delta*w, first_day+mtg_delta+week_delta*w) for w in range(weeks)]
weekly_meetings = [[c1,c2] for c1,c2 in possible if not(day_off(c1,no_class_ranges))]
meetings = [m for w in weekly_meetings for m in w]
meetings_string = [m.isoformat() for m in meetings]
weekly_meetings
```

Skip to main content

```
# possible = [(first_day+week_delta*w, first_day+mtg_delta+week_delta*w) for w in range(weeks)]
# weekly_meetings = [[c1,c2] for c1,c2 in possible if not(during_sb(c1))]
meetings = [m for w in weekly_meetings for m in w if not(m in skips)]


# build a table for the dates
badge_types = ['experience', 'review', 'practice']
target_cols = ['review_target','practice_target']
df_cols = badge_types + target_cols
badge_target_df = pd.DataFrame(index=meetings, data=[['future']*len(df_cols)]*len(meetings),
                               columns=df_cols).reset_index().rename(
                                  columns={'index': 'date'})
# set relative dates
today = date.today()
start_deadline = date.today() - timedelta(7)
complete_deadline = date.today() - timedelta(14)

# mark eligible experience badges
badge_target_df['experience'][badge_target_df['date'] <= today] = 'eligible'
# mark targets, cascading from most recent to oldest to not have to check < and >
badge_target_df['review_target'][badge_target_df['date'] <= today] = 'active'
badge_target_df['practice_target'][badge_target_df['date'] <= today] = 'active'
badge_target_df['review_target'][badge_target_df['date']
                         <= start_deadline] = 'started'
badge_target_df['practice_target'][badge_target_df['date']
                           <= start_deadline] = 'started'
badge_target_df['review_target'][badge_target_df['date']
                         <= complete_deadline] = 'completed'
badge_target_df['practice_target'][badge_target_df['date']
                           <= complete_deadline] = 'completed'
# mark enforced deadlines
badge_target_df['review'][badge_target_df['date'] <= today] = 'active'
badge_target_df['practice'][badge_target_df['date'] <= today] = 'active'
badge_target_df['review'][badge_target_df['date']
                         <= start_deadline] = 'started'
badge_target_df['practice'][badge_target_df['date']
                           <= start_deadline] = 'started'
badge_target_df['review'][badge_target_df['date']
                         <= complete_deadline] = 'completed'
badge_target_df['practice'][badge_target_df['date']
                           <= complete_deadline] = 'completed'
badge_target_df['review'][badge_target_df['date']
                         <= penalty_free_end] = 'penalty free'
badge_target_df['practice'][badge_target_df['date']
                         <= penalty_free_end] = 'penalty free'

# convert to numbers and set dates as index for heatmap compatibility
status_numbers_hm = {status:i+1 for i,status in enumerate(['future','eligible','active','penalty free','star
badge_target_df_hm = badge_target_df.replace(status_numbers_hm).set_index('date')

# set column names to shorter ones to fit better
badge_target_df_hm = badge_target_df_hm.rename(columns={'review':'review(e)','practice':'practice(e)',
                                                        'review_target':'review(t)','practice_target':'practi
# build a custom color bar
n_statuses = len(status_numbers_hm.keys())
manual_palette = [sns.color_palette("pastel", 10)[7],
                  sns.color_palette("colorblind", 10)[2],
                  sns.color_palette("muted", 10)[2],
                  sns.color_palette("colorblind", 10)[9],
                  sns.color_palette("colorblind", 10)[8],
                  sns.color_palette("colorblind", 10)[3]]
# generate the figure, with the colorbar and spacing
ax = sns.heatmap(badge_target_df_hm,cmap=manual_palette,linewidths=1)
# mote titles from bottom tot op
ax.xaxis.tick_top()
# pull the colorbar object for handling
colorbar = ax.collections[0].colorbar
# fix the location fo the labels on the colorbar
r = colorbar.vmax - colorbar.vmin
colorbar.set_ticks([colorbar.vmin + r / n_statuses * (0.5 + i) for i in range(n_statuses)])
colorbar.set_ticklabels(list(status_numbers_hm.keys()))
# add a title
```

Skip to main content

```
glue( 'today_notdisplayed' , not today ,display=false)
ax.set_title('Badge Status as of '+ today_string);
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[1], line 65
     60 mtg_delta = timedelta(meeting_days[1]-meeting_days[0])
     61 week_delta = timedelta(7)
---> 65 possible = [(first_day+week_delta*w, first_day+mtg_delta+week_delta*w) for w in range(weeks)]
     66 weekly_meetings = [[c1,c2] for c1,c2 in possible if not(day_off(c1,no_class_ranges))]
     67 meetings = [m for w in weekly_meetings for m in w]

NameError: name 'weeks' is not defined
```

Deadlines

We do not have a final exam, but URI assigns an exam time for every class. The date of that assigned exam will be the final due date for all work including all revisions.

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 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 (2pm) 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.

Skip to main content

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.

Expired badges will receive a comment and be closed

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 5 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. At most, you will receive feedback for one explore badge per week, so in order to earn six, your first one must be complete by March 18.

Build Badges

At most one build badge will be evaluated every 4 weeks. This means that if you want to earn 3 build badges, the first one must be in 8 weeks before the end of the semester, March 4. The second would be due April 1st, and the third submitted by the end of classes, April 29th.

Prepare work and Experience Badges Process

This is for a single example with specific dates, but it is similar for all future dates

The columns (and purple boxes) correspond to branches in your KWL repo and the yellow boxes are the things that you have to do. The “critical” box is what you have to wait for us on. The arrows represent PRs (or a local merge for the first one)

sequenceDiagram participant P as prepare Sep 12 participant E as experience Sep 12 participant M as main note over P: complete prepare work
 between feb Sep 7 and Sep12 note over E: run experience badge workflow <br/> at the end of class Sep12 P ->> E: local merge or PR you that <br/> does not need approval note over E: fill in experience reflection critical Badge review by instructor or TA E ->> M: Experience badge PR option if edits requested note over E: make requested edits option when approved note over M: merge badge PR end

In the end the commit sequence for this will look like the following:

Skip to main content

gitGraph commit commit checkout main branch prepare-2023-09-12 checkout prepare-2023-09-12 commit id: "gitunderstanding.md" branch experience-2023-09-12 checkout experience-2023-09-12 commit id: "initexp" merge prepare-2023-09-12 commit id: "fillinexp" commit id: "revisions" tag:"approved" checkout main merge experience-2023-09-12
Where the “approved” tag represents and approving reivew on the PR.

Review and Practice Badge

Legend:

flowchart TD badgestatus[[Badge Status]] passive[/ something that has to occur
 not done by student /] student[Something for you to do] style badgestatus fill:#2cf decisionnode{Decision/if} sta[action a] stb[action b] decisionnode --> |condition a|sta decisionnode --> |condition b|stb subgraph phase[Phase] st[step in phase] end
This is the general process for review and practice badges

flowchart TD %% subgraph work[Steps to complete] subgraph posting[Dr Brown will post the Badge] direction TB write[/Dr Brown finalizes tasks after class/] post[/Dr. Brown pushes to github/] link[/notes are posted with badge steps/] posted[[Posted: on badge date]] write -->post post -->link post --o posted end subgraph planning[Plan the badge] direction TB create[/Dr Brown runs your workflow/] decide{Do you need this badge?} close[close the issue] branch[create branch] planned[[Planned: on badge date]] create -->decide decide -->|no| close decide -->|yes| branch create --o planned end subgraph work[Work on the badge] direction TB start[do one task] commit[commit work to the branch] moretasks[complete the other tasks] ccommit[commit them to the branch] reqreview[request a review] started[[Started <br/> due within one week <br/> of posted date]] completed[[Completed
due within two weeks <br/> of posted date]] wait[/wait for feedback/] start --> commit commit -->moretasks commit --o started moretasks -->ccommit ccommit -->reqreview reqreview --> wait reqreview --o completed end subgraph review[Revise your completed badges] direction TB prreview[Read review feedback] approvedq{what type of review} merge[Merge the PR] edit[complete requested edits] earned[[Earned <br/> due by final grading]] discuss[reply to comments] prreview -->approvedq approvedq -->|changes requested|edit edit -->|last date to edit: May 1| prreview approvedq -->|comment|discuss discuss -->prreview approvedq -->|approved|merge merge --o earned end posting ==> planning planning ==> work work ==> review %% styling style earned fill:#2cf style completed fill:#2cf style started fill:#2cf style posted fill:#2cf style planned fill:#2cf

Explore Badges

flowchart TD subgraph proposal[Propose the Topic and Product] issue[create an issue] proposed[[Proposed]] reqproposalreview[Assign it to Dr. Brown] waitp[/wait for feedback/] proceedcheck{Did Dr. Brown apply a proceed label?} branch[start a branch] progress[[In Progress]] iterate[reply to comments and revise] issue --> reqproposalreview reqproposalreview --> waitp reqproposalreview --> proposed waitp --> proceedcheck proceedcheck -->|no| iterate proceedcheck -->|yes| branch branch --> progress iterate -->waitp end subgraph work[Work on the badge] direction TB moretasks[complete the work] ccommit[commit work to the branch] reqreview[request a review] wait[/wait for feedback/] complete[[Complete]] moretasks -->ccommit ccommit -->reqreview reqreview --o complete reqreview --> wait end subgraph review[Revise your work] direction TB prreview[Read review feedback] approvedq{what type of review} revision[[In revision]] merge[Merge the PR] edit[complete requested edits] earned[[Earned <br/> due by final grading]] prreview -->approvedq approvedq -->|changes requested|edit edit --> prreview edit --o revision approvedq -->|approved| merge merge --o earned end proposal ==> work work ==> review %% styling style proposed fill:#2cf style progress fill:#2cf style complete fill:#2cf style revision fill:#2cf style earned fill:#2cf

Build Badges

Skip to main content

reqproposalreview[Assign it] waitp[/wait for feedback/] proceedcheck{Did Dr. Brown apply a proceed label?} branch[start a branch]

progress[[In Progress]] iterate[reply to comments and revise] issue --> reqproposalreview reqproposalreview --> waitp reqproposalreview --> proposed waitp --> proceedcheck proceedcheck -->|no| iterate proceedcheck -->|yes| branch branch --> progress iterate -->waitp end subgraph work[Work on the badge] direction TB commit[commit work to the branch] moretasks[complete the work] draftpr[Open a draft PR and <br/> request a review] ccommit[incorporate feedback] reqreview[request a review] wait[/wait for feedback/] complete[[Complete]] commit -->moretasks commit -->draftpr draftpr -->ccommit moretasks -->reqreview ccommit -->reqreview reqreview --> complete reqreview --> wait end subgraph review[Revise your work] direction TB prreview[Read review feedback] approvedq{what type of review} revision[[In revision]] merge[Merge the PR] edit[complete requested edits] earned[[Earned <br/> due by final grading]] prreview -->approvedq approvedq -->|changes requested|edit edit --> prreview edit -->revision approvedq -->|approved| merge merge --o earned end proposal ==> work work ==> review %% styling style proposed fill:#2cf style progress fill:#2cf style complete fill:#2cf style revision fill:#2cf style earned fill:#2cf

Community Badges

These are the instructions from your `community_contributions.md` file in your KWL repo: For each one:

- In the `community_contributions.md`` file on your kwl repo, add an item in a bulleted list (start the line with -)
- Include a link to your contribution like `[text to display](url/of/contribution)`
- create an individual pull request titled “Community-shortname” where `shortname` is a short name for what you did. approval on this PR by Dr. Brown will constitute credit for your grade
- request a review on that PR from @brownsarahm

> **Important**
>
> You want one contribution per PR` for tracking

flowchart TD contribute[Make a contribution
 typically not in your KWL] link[Add a link to your <br> contribution to your
 communiyt_contribution.md
 in your KWL repo] pr[create a PR for that link] rev[request a review <br> from @brownsarahm] contribute --> link link --> pr --> rev rev --> approved[Dr. Brown approves] --> merge[Merge the PR] merge --o earned

Detailed Grade Calculations

> **Important**
>
> This page is generated with code and calculations, you can view them for more precise implementations of what the English sentences mean.

> **Warning**
>
> Some phrasing on this may change, but the core of what is required will not change

▶ Show code cell source

> **Tip**
>
> You c
> see m

Skip to main content

```
{'D ': 106,
 'D+': 124,
 'C-': 142,
 'C ': 192,
 'C+': 210,
 'B-': 228,
 'B ': 246,
 'B+': 264,
 'A-': 282,
 'A ': 300}
```

▶ Show code cell source

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[2], line 23
     21 learning_df = pd.Series(learning_weights,name ='complexity').reset_index()
     22 learning_df['badge_type'] = 'learning'
---> 23 comm_df  = pd.Series(community_weights,name='weight').reset_index()
     24 comm_df['badge_type'] = 'community'
     25 # nans are for learning badges which all ahve weight 1

NameError: name 'community_weights' is not defined
```

The total influence of each badge on the grade is as follows:

▶ Show code cell source

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[3], line 2
      1 # display
----> 2 influence_df[['badge_type','badge','complexity','weight','influence']]

NameError: name 'influence_df' is not defined
```

The total influence of a badge on your grade is the product of the badge's complexity. All learning badges have a weight of 1, but have varying complexity.

Bonuses

In addition to the weights for each badge, there also bonuses that will automatically applied to your grade at the end of the semester. These are for longer term patterns, not specific assignments. You earn these while workng on other assignments, not separately.

Important

the grade plans on the grading page and the thresholds above assume you earn the Participation and Lab bonuses for all grades a D or above and the Breadth bonus for all grades above a C.

Name	Definition	Influence	type
Participation	22 experience badges	18	auto
Lab	13 lab badges	18	auto
Breadth	If review + practice badges >-18:	32	auto
Git-ing unstuck	fix large mistakes your repo using advanced git operations and submit a short reflection (allowable twice; Dr. Brown must approve)	9	event
Early bird	6 review + practice submitted by 2/15	9	event
Descriptive commits	all commits in KWL repo and build repos after penalty free zone have descriptive commit messages (not GitHub default or nonsense)	9	event
Community Star	10 community badges	18	auto

Auto bonuses will be calculated from your other list of badges. Event bonuses will be logged in your KWL repo, where you get instructions when you meet the criteria.

Note

These bonuses are not pro-rated, you must fulfill the whole requirement to get the bonus. Except where noted, each bonus may only be earned once

Note

You cannot guarantee you will earn the Git-ing unstuck bonus, if you want to intentionally explore advanced operations, you can propose an explore badge, which is also worth 9.

Grade thresholds

Grade cutoffs for total influence are:

▶ Show code cell source

letter	threshold
D	106
D+	124
C-	142
C	192
C+	210
B-	228
B	246
B+	264
A-	282
A	300

Bonus Implications

Attendance and participation is *very* important:

- 14 experience, 6 labs, and 9 practice is an F
- 22 experience, 13 labs, and 9 practice is a C-
- 14 experience, 6 labs, 9 practice and one build is a C-
- 22 experience, 13 labs, 9 practice and one build is a C+

Missing one thing can have a nonlinear effect on your grade. Example 1:

- 22 experience, 13 labs, and 18 review is a C
- 21 experience, 13 labs, and 18 review is a C-
- 21 experience, 13 labs, and 17 review is a D+
- 21 experience, 12 labs, and 17 review is a D

Example 2:

- 22 experience, 13 labs, and 17 practice is a C
- 22 experience, 13 labs, 17 practice, and 1 review is a B-
- 22 experience, 13 labs, and 18 practice is a B

The Early Bird and Descriptive Commits bonuses are straight forward and set you up for success. Combined, they are also the same amount as the participation and lab bonuses, so getting a strong start and being detail oriented all semester can give you flexibility on attendance or labs.

Early Bird, Descriptive commits, Community Star, and Git-ing Unstuck are all equal to the half differnce between steps at a C or above. So earning any two can add a + to a C or a B for example:

- 22 experience, 13 labs, 18 practice, Descriptive Commits, and Early Bird is a B+

Skip to main content

in these two examples, doing the work at the start of the semester on time and being attentive throughout increases the grade without any extra work!

If you are missing learning badges required to get to a bonus, community badges will fill in for those first. If you earn the Participation, Lab, and Breadth bonuses, then remaining community badges will count toward the community bonus.

For example, at the end of the semester, you might be able to skip some the low complexity learning badges (experience, review, practice) and focus on your high complexity ones to ensure you get an A.

The order of application for community badges:

- to make up missing experience badges
- to make up for missing review or practice badges to earn the breadth bonus
- to upgrade review to practice to meet a threshold
- toward the community badge bonus

To calculate your final grade at the end of the semester, a script will count your badges and logged event bonuses. The script will output a dictionary with keys for each type of learning badge and event bonus with a count for the value of each.

```
example_student = {'experience' :21, 'lab': 13, 'review': 6,'practice': 12,
                   'explore': 2,
                   'build' :1,
                  'community': 3,
                  'unstuck': 0,
                  'descriptive': 1,
                  'early': 1 }
```

Then these counts will go into the following function to calculate the final grade

> **Warning**
>
> This is not complete, but will be before the end of the penalty free zone.

```
# set up remaining constants (some are above)
bonus_criteria = {'participation_bonus': lambda r: int(r['experience'] >=22),
                  'lab_bonus':  lambda r: int(r['lab'] >=13),
                   'breadth_bonus': lambda r: int(r['review'] + r['practice']>=18),
                 'community_bonus': lambda r: int(r['community']>=10),
                 'unstuck_bonus': lambda r: r['unstuck'],
                 'descriptive_bonus': lambda r: r['descriptive'],
                 'early_bonus': lambda r: r['early'] }
bonus_values = {'participation_bonus': bonus_participation,
                  'lab_bonus':  bonus_lab,
                   'breadth_bonus': bonus_breadth,
                 'community_bonus': 18,
                 'unstuck_bonus': 9,
                 'descriptive_bonus': 9,
                 'early_bonus': 9 }
weights = learning_weights.copy()
weights.update(bonus_values)
community_thresh = {'experience':22,
                 'review':4,
                 'practice':7,
                 'review_upgrade':3}
community_cost = {'experience':3,
                 'review':4,
                 'practice':7,
                 'review_upgrade':3}

# compute grade

# def calculate_grade(badge_dict):
#     if badge_dict['community']>0:
        # apply community badges

    # add bonuses


    # final sum
    # sum([weights[cat]*count_dict[cat] for cat in count_dict.keys()])
```

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?

Skip to main content

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

Recommended workload distribution

> **Note**
>
> General badge deadlines are on the detailed badge procedures page.

To plan your time, I recommend expecting the following:

- 30 minutes, twice per week for prepare work (typically not this much).
- 1.5(review)-3(practice) hours, twice per week for the dated badges (including revisions).

For each explore:

- 30 min for proposal
- 7 hours for the project

For each build:

Skip to main content

- 22 hours for the project
- 30 min for the final reflection

This is a four credit course, meaning we have approximately 4 hours of class + lab time per week($75 \times 2 + 105 = 255$ minutes or 4.25 hours). By the accredidation standards, students should spend a minimum of 2 hours per credit of work outside of class over 14 weeks. For a 4 credit class, then, the expected minimum number of hours of work outside of class you should be spending is 112 hours(2 * 4 * 14). With these calculations, given that there are 26 class sessions and only 18 review or practice are required, it is possible to earn an A with approximately 112 hours of work outside of class and lab time.

Tentative Timeline

> **Warning**
>
> This section is not yet updated for spring 2024.
>
> This is a rough example.

This is the planned schedule, but is subject to change in order to adapt to how things go in class or additional questions that come up.

```
import pandas as pd
pd.read_csv('schedule.csv',index_col='date').sort_index()
```

date	question	keyword	conceptual	practical	social	activity
2023-09-07	Welcome, Introduction, and Setup	intro	what is a system, why study tools	GitHub basics	class intros	create kwl repo in github, navigate github.com...
2023-09-12	Course Logistics and Learning	logistics	github flow with issues	syllabus	working together and building common vocab	set up to work offline together, create a folder
2023-09-14	Bash intro & git offline	terminal start	git structure, paths and file system	bash path navigation, git terminal authentication	why developers work differently than casual users	navigate files and clone a repo locally
2023-09-19	How can I work with branches offline?	gitoffline	git branches	github flow offline, resolving merge conflicts	commuication is important, git can help fix mi...	clone a repo and make a branch locally
2023-09-21	When do I get an advantage from git and bash?	why terminal	computing mental model, paths and file structure	bash navigation, tab completion	collaboration requires shared language, shared...	work with bash and recover from a mistake with...
2023-09-26	What *is* a commit?	merge conflicts	versions, git vlaues	merge conflicts in github, merge conflicts wit...	human and machine readable, commit messages ar...	examine commit objects, introduce plumbing com...
2023-09-28	How do programmers communicate about code?	documentation	build, automation, modularity, pattern matching,	generate documentation with jupyterbook, gitig...	main vs master, documentation community	make a jupyterbook
2023-10-03	What *is* git?	git structure	what is a file system, how does git keep track...	find in bash, seeing git config, plumbing/porc...	git workflows are conventions, git can be used...	examine git from multiple definitions and insp...
2023-10-05	Why are these tools like this?	unix philosophy	unix philosophy, debugging strategies	decision making for branches	social advantages of shared mental model, diff...	discussion with minor code examples
2023-10-12	How does git make a commit?	git internals	pointers, design and abstraction, intermediate...	inspecting git objects, when hashes are unique...	conventions vs requirements	create a commit using plumbing commands
2023-10-17	What is a commit number?	numbers	hashes, number systems	git commit numbers, manual hashing with git	number systems are derived in culture	discussion and use hashing algorithm
2023-10-19	How can can I release and share my code?	git references	pointers, git branches and tags	git branches, advanced fixing, semver and conv...	advantages of data that is both human and mach...	make a tag and release
2023-10-24	How can I automate things with bash?	bash scripting	bash is a programming language, official docs,...	script files, man pages, bash variables, bash ...	using automation to make collaboration easier	build a bash script that calculates a grade
2023-10-26	How can I work on a remote server?	server	server, hpc, large files	ssh, large files, bash head, grep,	hidden impacts of remote	log into a remote server and work

Skip to main content

date	question	keyword	conceptual	practical	social	activity
2023-10-31	What is an IDE?	IDE	IDE parts	compare and contrast IDEs	collaboraiton features, developer communities	discussions and sharing IDE tips
2023-11-02	How do I choose a Programming Language for a p...	programming languages	types of PLs, what is PL studying	choosing a language for a project	usability depends on prior experience	discussion or independent research
2023-11-07	How can I authenitcate more securely from a te...	server use	ssh keys, hpc system strucutre	ssh keys, interactive, slurm	social aspects of passwords and security	configure and use ssh keys on a hpc
2023-11-09	What Happens when we build code?	building	building C code	ssh keys, gcc compiler	file extensions are for people, when vocabular...	build code in C and examine intermediate outputs
2023-11-14	What happens when we run code?	hardwar	von neuman architecture	reading a basic assembly language	historical context of computer architecures	use a hardware simulator to see step by step o...
2023-11-16	How does a computer represent non integer quan...	floats	float representation	floats do not equal themselves	social processes around standard developents, ...	work with float representation through fractio...
2023-11-21	How can we use logical operations?	bitwise operation	what is a bit, what is a register, how to brea...	how an ALU works	tech interviews look for obscure details somet...	derive addition from basic logic operations
2023-11-28	What *is* a computer?	architecture	physical gates, history	interpretting specs	social context influences technology	discussion
2023-11-30	How does timing work in a computer?	timing	timing, control unit, threading	threaded program with a race condition	different times matter in different cases	write a threaded program and fix a race condition
2023-12-05	How do different types of storage work together?	memory	different type of memory, different abstractions	working with large data	privacy/respect for data	large data that has to be read in batches
2023-12-07	How does this all work together	review	all	end of semester logistics	group work final	review quiz, integration/reflection questions
2023-12-12	How did this semester go?	feedback	all	grading	how to learn better together	discussion

Tentative Lab schedule

```
pd.read_csv('labschedule.csv',index_col='date').sort_index()
```

Skip to main content

date	topic	activity
2023-09-08	GitHub Basics	syllabus quiz, setup
2023-09-15	working at the terminal	organization, setup kwl locally, manage issues
2023-09-22	offline branches	plan for success, clean a messy repo
2023-09-29	tool familiarity	work on badges, self progress report
2023-10-06	unix philosophy	design a command line tool that would enable a...
2023-10-13	git plumbing	git plumbing experiment
2023-10-20	git plumbing	grade calculation script, self reflection
2023-10-27	scripting	releases and packaging
2023-11-03	remote, hpc	server work, batch scripts
2023-11-10	Compiling	C compiling experiments
2023-11-17	Machine representation	bits and floats and number libraries
2023-12-01	hardware	self-reflection, work, project consultations
2023-12-08	os	hardware simulation

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.

Skip to main content

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 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, Access, and Inclusion Services for Students Statement

Your access in this course is important. Please send me your Disability, Access, and Inclusion (DAI) 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 DAI, please contact them to engage in a confidential conversation about the process for requesting reasonable accommodations in the classroom. DAI can be reached by calling: 401-874-2098, visiting: web.uri.edu/disability, or emailing: dai@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 or email address associated with a commit 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

> **Tip**
>
> Assignments are tested against LLMs and designed so that they require you to learn and that an LLM answer will be low quality and not earn credit.

All of your work must reflect your own thinking and understanding. The written work in English that you submit must all be your own work or content that was provided to you in class, it cannot include text that was generated by an AI or plagiarized in any other way. You may use auto-complete in all tools incuding, IDE-integrated GitHub co-pilot (or similar, IDE embedded tool) for any code that is required for this course because the code is necessary to demonstrate examples, but language syntax is not the core learning outcome.

If you are found to submit prismia responses that do not reflect your own thinking or that of discussion with peers as directed, the experience badge for that class session will be ineligible.

If work is suspected to be the result of inappropriate collaboration or AI use, you will be allowed to take an oral exam in lab time to contest and prove that your work reflects your own understanding.

The first time you will be allowed to appeal through an oral exam. If your appeal is successful, your counter resets. If you are found to have violated the policy then the badge in question will be ineligible and your maximum number of badges possible to be earned will be limited according to the guidelines below per badge type (you cannot treat the plagiarized badge as skipped). If you are found to have violated the policy a second time, then no further work will be graded for the remainder of the semester.

If you are found to submit work that is not your own for a *review or practice* badge, the review and practice badges for that date will be ineligible and the penalty free zone terms will no longer apply to the first six badges.

If you are found to submit work that is not your own for an *explore or build* badge, that badge will not be awarded and your maximum badges at the level possible will drop to 2/3 of the maximum possible.

Viral Illness Precautions

The University is committed to delivering its educational mission while protecting the health and safety of our community. Students who are experiencing symptoms of viral illness should NOT go to class/work. Those who test positive for COVID-19 should follow the isolation guidelines from the Rhode Island Department of Health and CDC.

If you miss class, you do not need to notify me in advance. You can follow the makeup procedures.

Excused Absences

Absences due to serious illness or traumatic loss, religious observances, or participation in a university sanctioned event are considered excused absences.

You do not need to notify me in advance.

For short absences (1-2 classes) can follow the makeup procedures.

For this, email Dr. Brown when you are ready to get caught up and she will help you make a plan for the best order to complete missed work so that you are able to participate in subsequent activities. Extensions on badges will be provided if needed. In your plan, include what class sessions you missed by date.

Skip to main content

Mental Health and Wellness:

We understand that college comes with challenges and stress associated with your courses, job/family responsibilities and personal life. URI offers students a range of services to support your mental health and wellbeing, including the URI Counseling Center, MySSP (Student Support Program) App, the Wellness Resource Center, and Well-being Coaching.

Office Hours & Communication

Announcements

Announcements will be made via GitHub Release. You can view them online in the releases page or you can get notifications by watching the repository, choosing “Releases” under custom see GitHub docs for instructions with screenshots. You can choose GitHub only or e-mail notificaiton from the notification settings page

> **Warning**
>
> For the first week they will be made by BrightSpace too, but after that, all course activities will be only on GitHub.

> **Sign up to watch**
>
> Watch the repo and then, after the first class, claim a community badge for doing so, using a link to these instructions as the “contribution” like follows.
>
> ```
> - [watched the repo as per announcements](https://introcompsys.github.io/spring2023/syllabus/communi
> ```
>
> put this on a branch called `watch_community_badge` and title your PR “Community-Watch”

Help Hours

Day	Time	Location	Host
Monday	1:00 - 3:00	139 Tyler	Marcin
Monday	4:45-6:30	052 Tyler	Trevor
TBD	TBD	Zoom	Dr. Brown
TBD	TBD	134 Tyler	Dr. Brown
Tuesday	11-12:30	052 Tyler	Trevor
Wednesday	10:00 - 12:00	139 Tyler	Marcin
Wednesday	2:00 - 4:00	049 Tyler	Gyanko
Thursday	10:00 - 11:00	Zoom	Marcin
Thursday	11-12:30	052 Tyler	Trevor
Friday	2:00 - 4:00	049 Tyler	Gyanko
TBD	TBD	Zoom	TA
TBD	TBD	in person TBD	TA

Online office hours locations are linked on the GitHub Organization Page

Important

You can only see them if you are a “member” to join, make sure that you have completed Lab 0.

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 isses

- 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 tpo or you find an additional helpful resource related to something)

...

You can submit a pull request for the typo above, but be sure to check the pull request tab of the repo before submitting

For E-mail

- use e-mail for general inquiries or notifications
- 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.

> **Should you e-mail your work?**
>
> No, request a pull request review or make an issue if you are stuck

KWL Chart

Working with your KWL Repo

> **Important**
>
> The `main` branch should only contain material that has been reviewed and approved by the instructors.

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.

> **Tip**
>
> You c
> on yo

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 | _ | _ | _ |
```

Skip to main content

```
|compiling | _ | _ | _ |
| linking   | _ | _ | _ |
| building | _ | _ | _ |
| machine representation  | _ | _ | _ |
| integers   | _ | _ | _ |
| floating point  | _ | _ | _ |
|logic gates | _ | _ | _ |
| ALU | _ | _ | _ |
| binary operations | _ | _ | _ |
| memory | _ | _ | _ |
| cache | _ | _ | _ |
| register | _ | _ | _ |
| clock | _ | _ | _ |
| Concurrency | _ | _ | _ |
```

Required Files

This lists the files for reference, but mostly you can keep track by badge issue checklists.

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.

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.

2023-09-07

related notes

Activities:

2023-09-12

related notes

Activities:

2023-09-14

related notes

Activities:

2023-09-19

related notes

Activities:

2023-09-21

related notes

Activities:

2023-09-26

related notes

Activities:

2023-09-28

related notes

Activities:

2023-10-03

related notes

Activities:

2023-10-05

related notes

Activities:

2023-10-12

related notes

Activities:

2023-10-17

Activities:

2023-10-19

related notes

Activities:

2023-10-24

related notes

Activities:

2023-10-31

related notes

Activities:

2023-11-02

related notes

Activities:

2023-11-07

related notes

Activities:

2023-11-14

related notes

Activities:

2023-11-16

related notes

Activities:

2023-11-21

related notes

Activities:

2023-11-28

related notes

Activities:

2023-11-30

related notes

Activities:

2023-12-05

related notes

Activities:

2023-12-07

related notes

Activities:

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. These are evaluated on completion only, but we may ask you questions or leave comments if appropriate, in that event you should reply and then we'll approve.

2023-09-12

Activities:

2023-09-14

2023-09-19

Activities:

2023-09-21

Activities:

2023-09-26

Activities:

2023-09-28

Activities:

2023-10-03

Activities:

2023-10-05

Activities:

2023-10-12

Activities:

2023-10-17

Activities:

2023-10-19

Activities:

2023-10-24

Activities:

Activities:

2023-10-31

Activities:

2023-11-02

Activities:

2023-11-02

related notes

Activities:

2023-11-07

related notes

Activities:

2023-11-14

related notes

Activities:

2023-11-16

related notes

Activities:

2023-11-21

related notes

Activities:

2023-11-28

related notes

Activities:

2023-11-30

related notes

Activities:

2023-12-05

related notes

Activities:

2023-12-07

related notes

Activities:

More Practice Badges

> **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.

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

2023-09-07

related notes

Activities:

2023-09-12

related notes

Activities:

2023-09-14

Activities:

2023-09-19

related notes

Activities:

2023-09-21

related notes

Activities:

2023-09-26

related notes

Activities:

2023-09-28

related notes

Activities:

2023-10-03

related notes

Activities:

2023-10-05

related notes

Activities:

2023-10-12

related notes

Activities:

2023-10-17

related notes

Activities:

2023-10-19

related notes

Activities:

2023-10-24

related notes

Activities:

2023-10-31

related notes

Activities:

2023-11-02

related notes

Activities:

2023-11-07

related notes

Activities:

2023-11-14

related notes

Activities:

2023-11-16

related notes

2023-11-21

related notes

Activities:

2023-11-28

related notes

Activities:

2023-11-30

related notes

Activities:

2023-12-05

related notes

Activities:

2023-12-07

related notes

Activities:

KWL File List

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?

Create an issue on your kwl repo, label it explore, and “assign” @brownsarahm.

In your issue, describe the question you want to answer or topic to explore and the format you want to use.

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)

Skip to main content

- 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
- Review a reference or resource for a topic
- write a code tour that orients a new contributor to a past project or an open source tool you like.

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.

Build Badges

Build may be individual or in pairs.

Proposal Template

If you have selected to do a project, please use the following template to propose a bulid

```
## < Project Tite >

<!-- insert a 1 sentence summary  -->

### Objectives

<!-- in this section describe the overall goals in terms of what you will learn and the problem you will solv

### Method

 <!-- describe what you will do , will it be research, write & present? will there be something you build? wi

### Deliverables

<!-- list what your project will produce with target deadlines for each-->

### Milestones
```

The deliverables will depend on what your method is, which depend on your goals. It must be approved and the final submitted will have to meet what is approved. Some guidance:

- any code or text should be managed with git (can be GitHub or elsewhere)
- if you write any code it should have documentation
- if you do experiments the results should be summrized
- if you are researching something, a report should be 2-4 pages, plus unlimited references in the 2 column ACM format.

This guidance is generative, not limiting, it is to give ideas, but not restrict what you *can* do.

Updates and work in Progress

Skip to main content

Summary Report

This summary report will be added to your kwl repo as a new file `build_report_title.md` where `title` is the (title or a shortened version) from the proposal.

This summary report have the following sections.

1. **Abstract** a one paragraph “abstract” type overview of what your project consists of. This should be written for a general audience, something that anyone who has taken up to 211 could understand. It should follow guidance of a scientific abstract.
2. **Reflection** a one paragraph reflection that summarizes challenges faced and what you learned doing your project
3. **Artifacts** links to other materials required for assessing the project. This can be a public facing web resource, a private repository, or a shared file on URI google Drive.

Collaborative Build rules/procedures

- Each student must submit a proposal PR for tracking purposes. The proposal may be shared text for most sections but the deliverables should indicate what each student will do (or be unique in each proposal).
- the proposal must indicate that it is a pair project, if iteration is required, I will put those comments on both repos but the students should discuss and reply/edit in collaboration
- the project must include code reviews as a part of the workflow links to the PRs on the project repo where the code reviews were completed should be included in the reflection
- each student must complete their own reflection. The abstract can be written together and shared, but the reflection must be unique.

Build Ideas

General ideas to write a proposal for

- make a vs code extension for this class or another URI CS course
- port the courseutils to rust. crate clap is like the python click package I used to develop the course utils
- buld a polished documentation website for your CSC212 project with sphinx or another static site generator
- use version control, including releases on any open source side-project and add good contributor guidelines, README, etc

Auto-approved proposals

For these build options, you can copy-paste the template below to create your proposal issue and assign it to `@brownsarahm`.

For working alone there are two options, for working with a partner there is one.

212 Project Solo- Docs focus

Use this option if your team for your 212 project is not currently enrolled in this class or does not want to do a collaborative build. This version focuses on the user docs.

Skip to main content

```
## 212 Project Doc & Developer onboarding

Add documentation website and developer onboarding information to your CSC 212 project.

### Objectives

<!-- in this section describe the overall goals in terms of what you will learn and the problem you will solv

This project will provide information for a user to use the data structure implemented for a CSC 212 project

### Method

 <!-- describe what you will do , will it be research, write & present? will there be something you build? wi
1. ensure there is API level documentation in the code files
1. build a documentation website using [jupyterbook/ sphinx/doxygen/] that includes setup instructions and ex
1. configure the repo to automatically build the documentation website each time the main branch is updated


### Deliverables


- link to repo with the contents listed in method in the reflection file


### Milestones

<!-- give a target timeline -->
```

212 Project Solo- Developer focus

Use this option if your team for your 212 project is not currently enrolled in this class or does not want to do a collaborative build. This version focuses on the contributor experience.

```
## 212 Project Doc & Developer onboarding

Add documentation website and developer onboarding information to your CSC 212 project.

### Objectives

<!-- in this section describe the overall goals in terms of what you will learn and the problem you will solv

This project will provide information for a user to use the data structure implemented for a CSC 212 project

### Method

 <!-- describe what you will do , will it be research, write & present? will there be something you build? wi

1. ensure there is API level documentation in the code files
1. add a license, readme, and contributor file
1. add [code tours](https://marketplace.visualstudio.com/items?itemName=vsls-contrib.codetour) that help some
1. set up a PR template
1. set up 2 issue templates: 1 for feature request and 1 for bug reporting


### Deliverables


- link to repo with the contents listed in method in the reflection file


### Milestones

<!-- give a target timeline -->
```

212 Project Pair

Use this option if your teammate for your 212 project is in this class and wants to do a collaborative build.

```
## 212 Project Doc & Developer onboarding

Add documentation website and developer onboarding information to your CSC 212 project.

### Objectives

<!-- in this section describe the overall goals in terms of what you will learn and the problem you will solv

This project will provide information for a user to use the data structure implemented for a CSC 212 project

### Method

 <!-- describe what you will do , will it be research, write & present? will there be something you build? wi
1. ensure there is API level documentation in the code files
1. build a documentation website using [jupyterbook/ sphinx/doxygen/] that includes setup instructions and ex
1. configure the repo to automatically build the documentation website each time the main branch is updated
1. add [code tours](https://marketplace.visualstudio.com/items?itemName=vsls-contrib.codetour) that help some
1. set up a PR template
1. set up 2 issue templates: 1 for feature request and 1 for bug reporting



### Deliverables


- link to repo with the contents listed in method in the reflection file


### Milestones

<!-- give a target timeline -->
```

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.

You will be able to use provided command line tools and github actions to produce a report of your status at any time from your PR list, starting in the third week. 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 help you create and update your success plan for the course.

Also, when are each badge due, time wise?

within 2 weeks, but again recommend finishing sooner.
Experience reports for missing class is on a case by case basis depending on why you missed class. You must have a plan by the next class.

Explore and build, we'll agree to a deadline when you propose.

Who should I request to review my work?

- Experience badge (inclass): TA in your group
- Experience report(makeup) `@brownsarahm``
- Review badge: `@instructors` team (it will convert to one of the three of us)
- Explore Proposal: `@brownsarahm`
- Explore Badge: `@brownsarahm`
- Build Proposal: `@brownsarahm`
- Build Badge: `@brownsarahm`

Will everything done in the penalty free zone be approved even if there are mistakes?

No. In the penalty-free zone I still want you to learn things, but we will do extra work to make sure that you get credit for all of your effort even if you make mistakes in how to use GitHub. We will ask you to fix things that we have taught you to fix, but not things that we will not cover until later.

The goal is to make things more fair while you get used to GitHub. It's a nontrivial thing to learn, but getting used to it is worth it.

I want this class to be a safe place for you to try things, make mistakes and learn from them without penalty. A job is a much higher stakes place to learn a tool as hard as GitHub, so I want this to be lower stakes, even though I cannot promise it will be easy.

Once we make revisions on a pull request, how do we notify you that we have done them?

You do not have to do anything, GitHub will automatically notify which ever one of us who reviewed it initially when you make changes.

What should work for an explore badge 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.

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

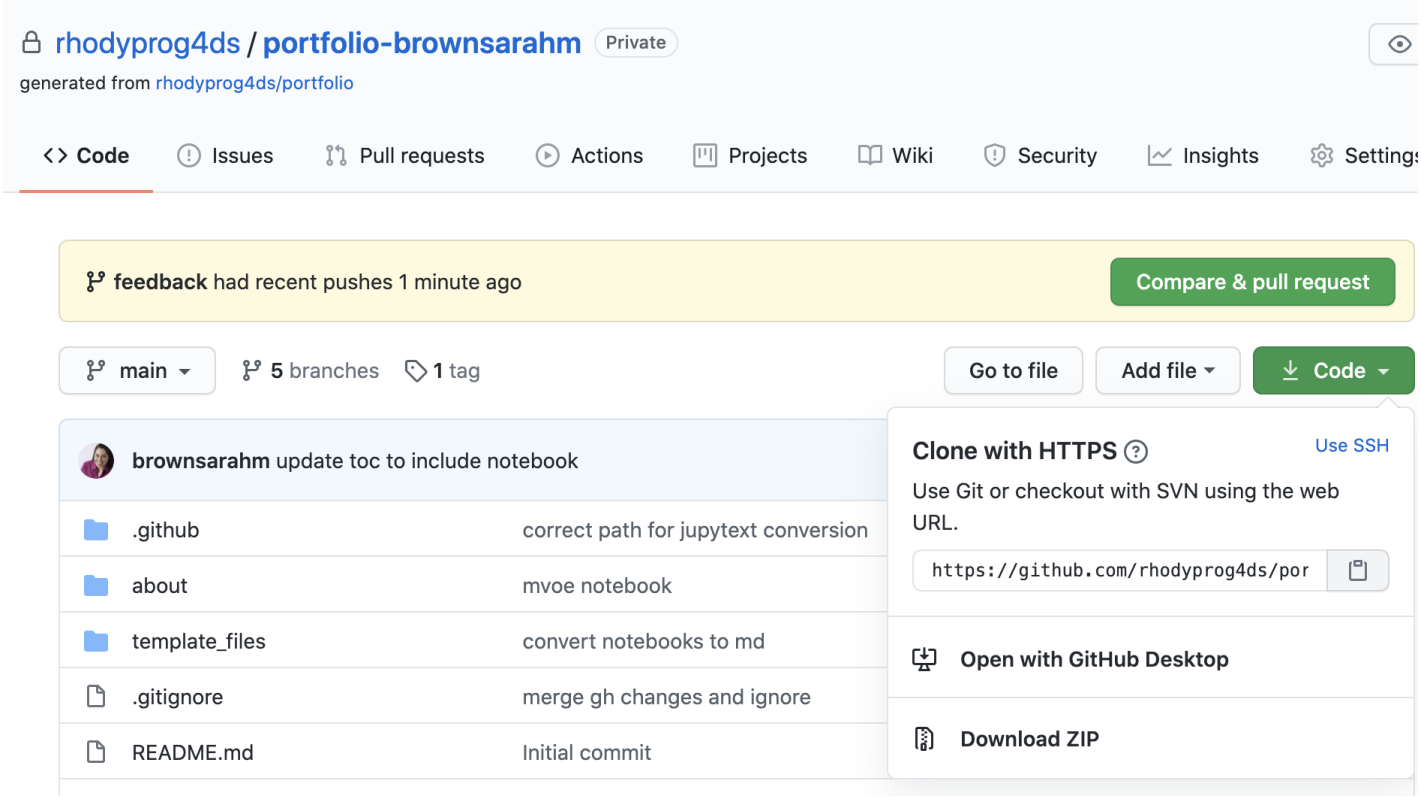
Help! I accidentally merged the Badge Pull Request before my assignment was graded

That’s ok. You can fix it.

note: these instructions use the main branch the way we use the badge branches and the feedback branch the way we use the main branch in this course

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



Next open a terminal or GitBash and type the following.

```
git clone
```

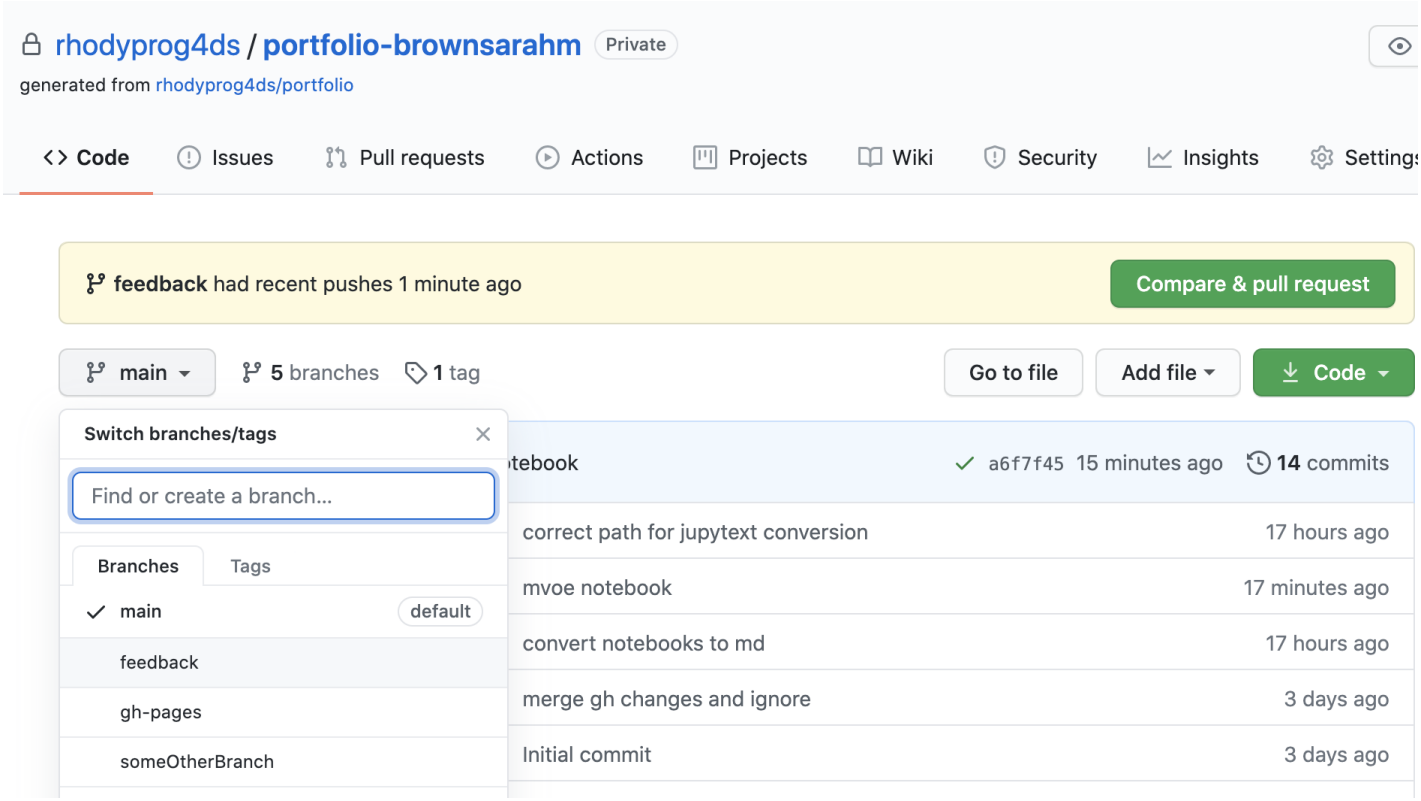
then past your url that you copied. It will look something like this, but the last part will be the current assignment repo and your username.

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

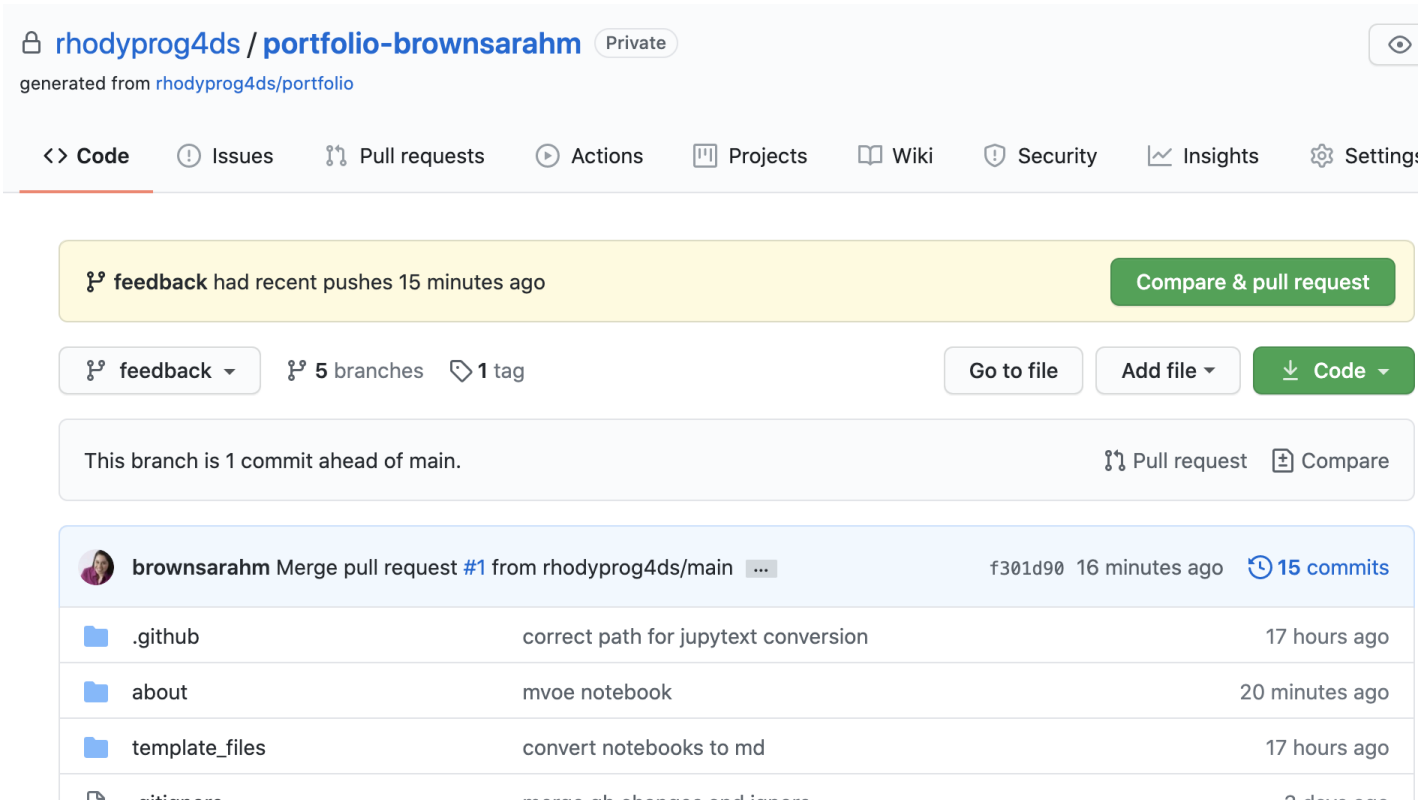
When you merged the Feedback pull request you advanced the `feedback` branch, so we need to hard reset it back to before you did any work. To do this, first check it out, by navigating into the folder for your repository (created when you cloned above) and then checking it out, and making sure it's up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

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

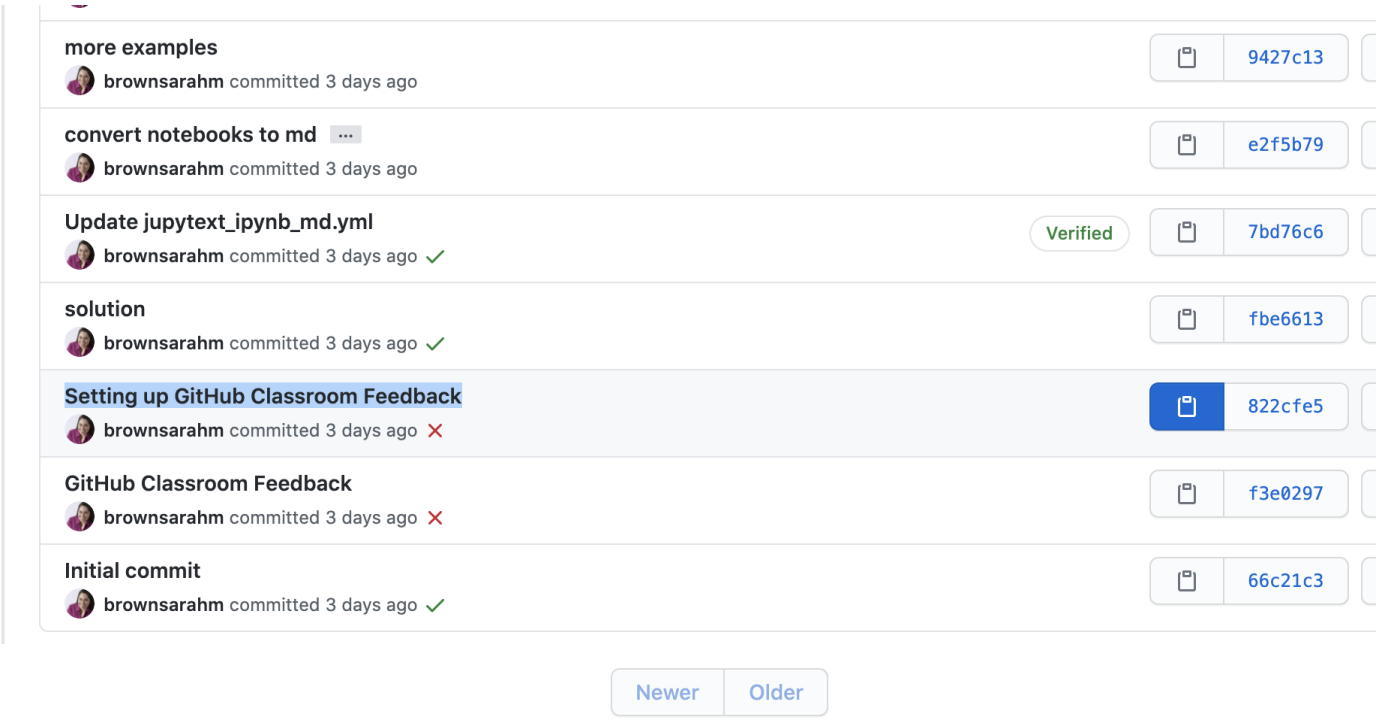
Now, you have to figure out what commit to revert to, so go back to GitHub in your browser, and swithc to the feedback branch there. Click on where it says `main` on the top right next to the branch icon and choose feedback from the list.



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



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

```
(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.

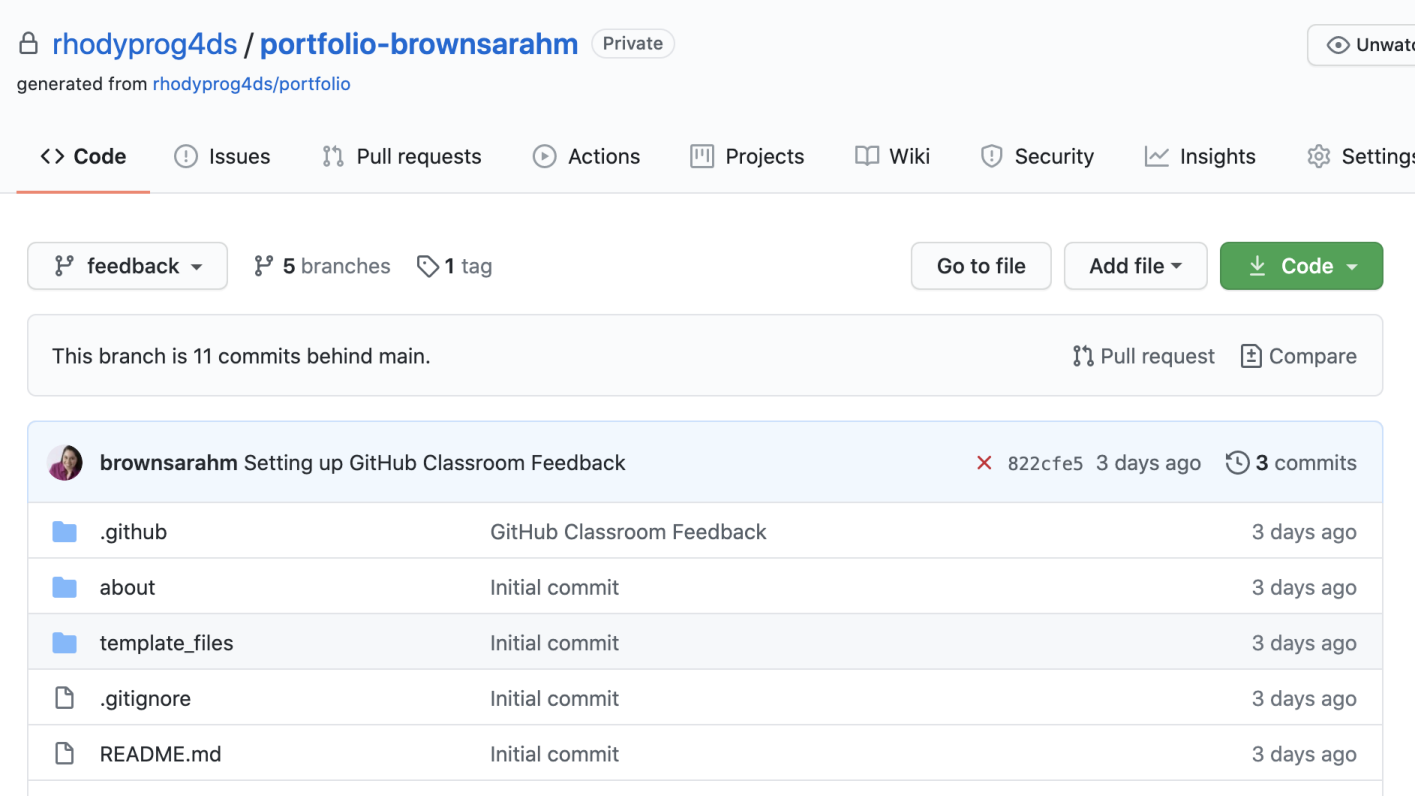
```
git push origin -f
```

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

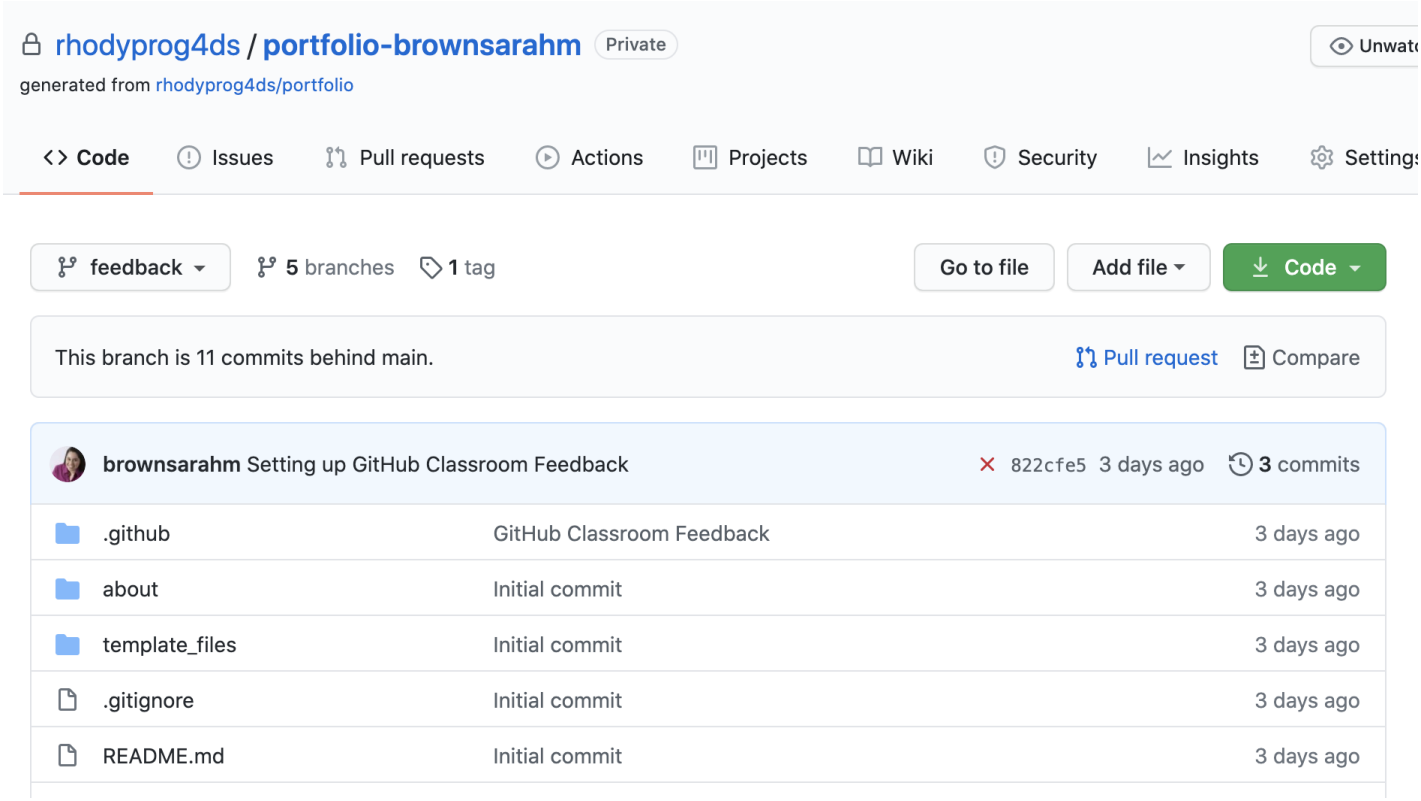
```
Total 0 (delta 0), reused 0 (delta 0)
To https://github.com/rhodyprog4ds/portfolio-brownsarahm.git
 + f301d90...822cfe5 feedback -> feedback (forced update)
```

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

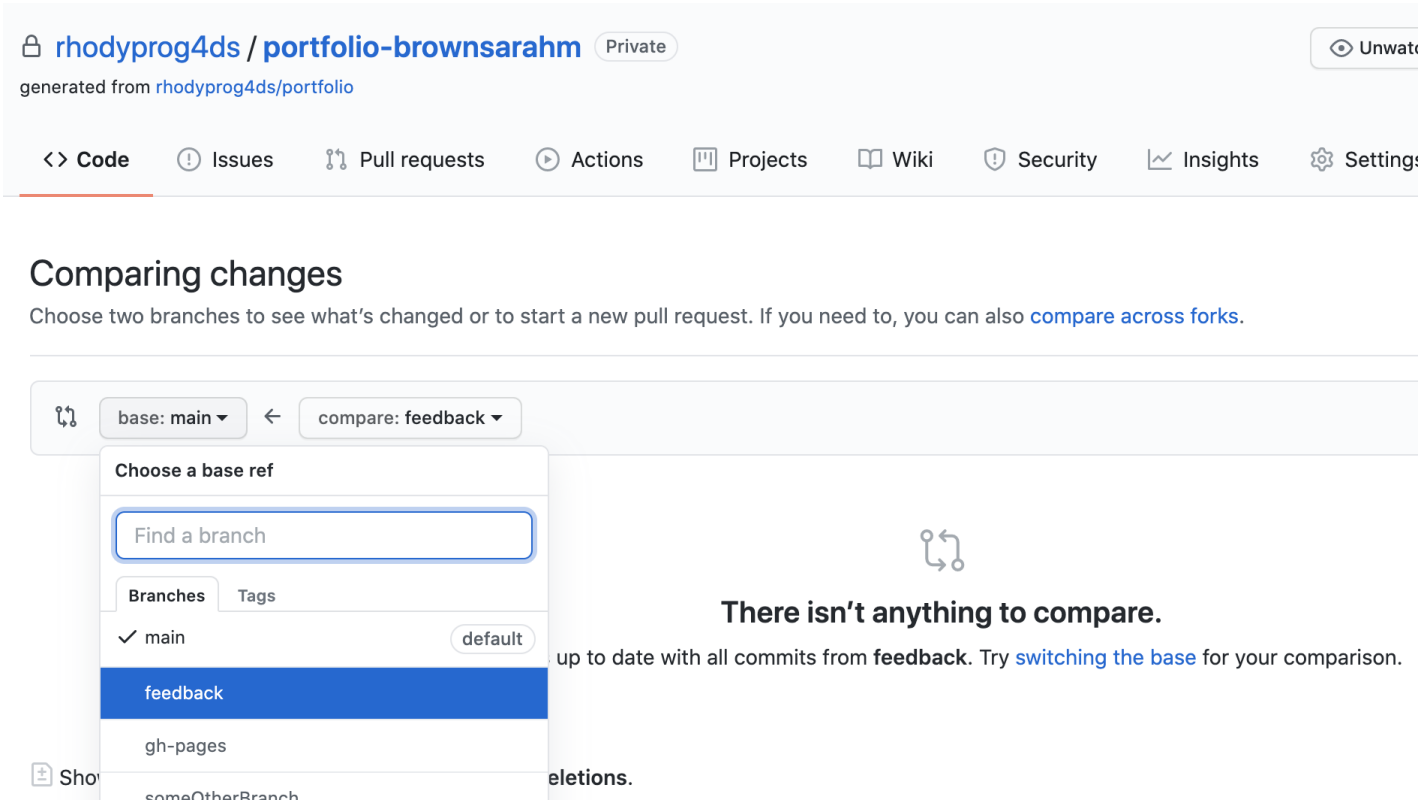
Now back on GitHub, in your browser, click on the code tab. It should look something like this now. Notice that it says, "This branch is 11 commits behind main" your number will be different but it should be 1 less than the number you had when you checked `git status`. This is because we reverted the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



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

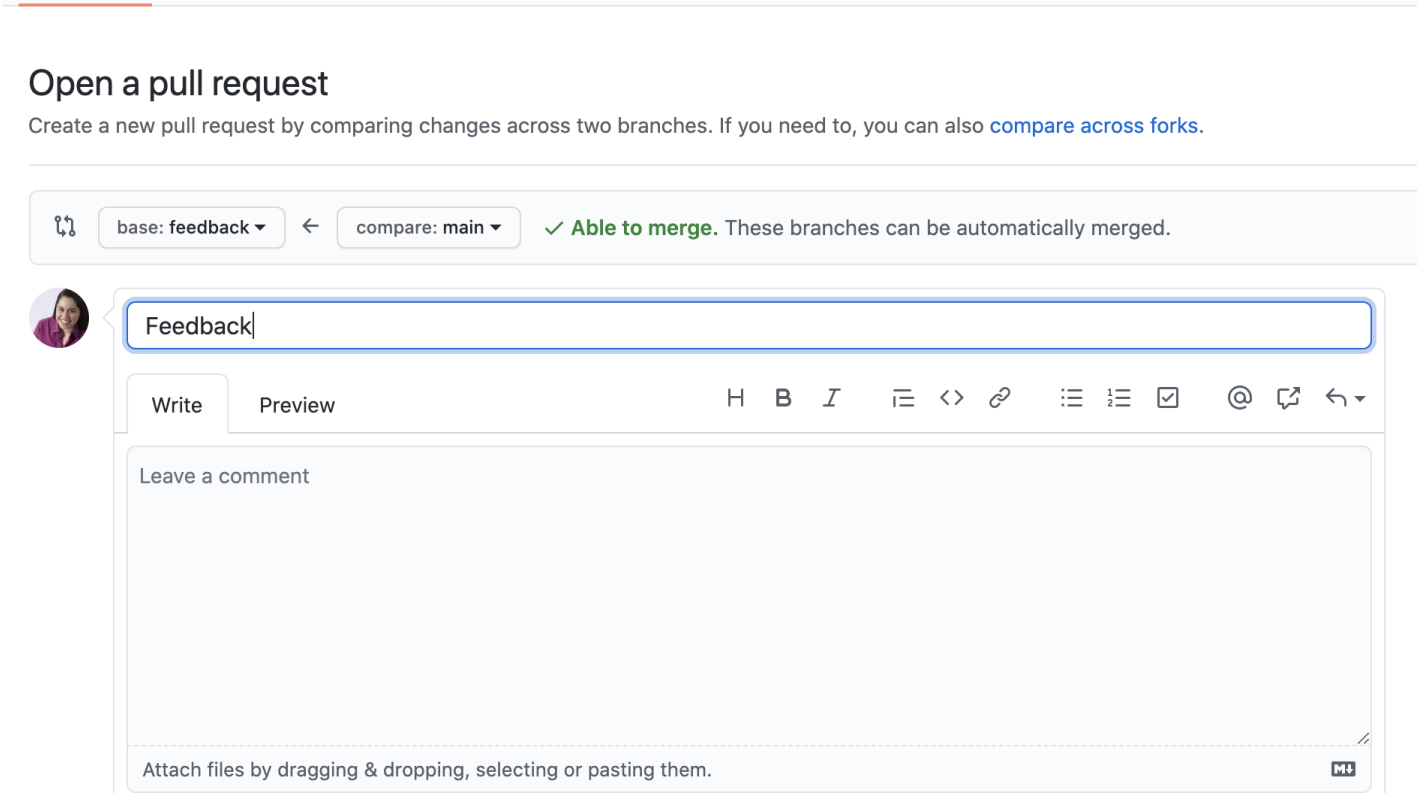


It will say there isn't anything to compare, but this is because it's trying to use `feedback` to update `main`. We want to use `main` to update `feedback` for this PR. So we have to swap them. Change base from `main` to `feedback` by clicking on it and choosing `feedback` from the list.



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

Skip to main content



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
```

Other Course Software/tools

Courseutils

This is how your badge issues are created. It also has some other utilities for the course. It is open source and questions/issues should be posted to its issue tracker

Jupyterbook

Glossary

> **Tip**
>
> Contributing glossary terms or linking to uses of glossary terms to this page is eligible for community badges

absolute path

the path defined from the root of the system

add (new files in a repository)

the step that stages/prepares files to be committed to a repository from a local branch

bash

bash or the bourne-again shell is the primary interface in UNIX based systems

bitwise operator

an operation that happens on a bit string (sequence of 1s and 0s). They are typically faster than operations on whole integers.

branch

a copy of the main branch (typically) where developmental changes occur. The changes do not affect other branches because it is isolated from other branches.

Compiled Code

code that is put through a compiler to turn it into lower level assemlby language before it is executed. must be compiled and re-executed everytime you make a change.

directory

a collection of files typically created for organizational purposes

fixed point number

the concept that the decimal point does not move in the number. Cannot represent as wide of a range of values as a floating point number.

floating point number

the concept that the decimal can move within the number (ex. scientific notation; you move the decimal based on the exponent on the 10). can represent more numbers than a fixed point number.

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

.gitignore

a file in a git repo that will not add the files that are included in this .gitignore file. Used to prevent files from being unnecessarily committed.

Skip to main content

git Plumbing commands

low level git commands that allow the user to access the inner workings of git.

git Workflow

a recipe or recommendation for how to use Git to accomplish work in a consistent and productive manner

HEAD

a file in the .git directory that indicates what is currently checked out (think of the current branch)

merge

putting two branches together so that you can access files in another branch that are not available in yours

mermaid

meraid syntax allows user to create precise, detailed diagrams in markdown files.

hash function

the actual function that does the hashing of the input (a key, an object, etc.)

hashing

putting an input through a function and unique fixed length output (the output is called a hash; used in hash tables and when git hashes commits).

integrated development environment

also known as an IDE, puts together all of the tools a developer would need to produce code (source code editor, debugger, ability to run code) into one application so that everything can be done in one place. can also have extra features such as showing your file tree and connecting to git and/or github.

interpreted code

code that is directly executed from a high level language. more expensive computationally because it cannot be optimized and therefore can be slower.

issue

provides the ability to easily track ideas, feedback, tasks, or bugs. branches can be created for specific issues. an issue is open when it is created. pull requests have the ability to close issues.

Linker

a program that links together the object files and libraries to output an executable file.

path

the “location” of a file or folder(directory) in a computer

pull (changes from a repository)

download changes from a remote repository and update the local repository with these changes.

pull request

allow other users to review and request changes on branches. after a pull request recieves approval you can merge the

Skip to main content

push (changes to a repository)

to put whatever you were working on from your local machine onto a remote copy of the repository in a version control system.

relative path

the path defined **relative** to another file or the current working directory; may start with a name, includes a single file name or may start with `./`

repository

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

ROM (Read-Only Memory)

Memory that only gets read by the CPU and is used for instructions

SHA 1

the hashing function that git uses to hash its functions (found to have very serious collisions (two different inputs have same hashes), so a lot of software is switching to SHA 256)

sh

abbr. see shell

shell

a command line interface; allows for access to an operating system

ssh

allows computers to safely connect to networks (such as when we used an ssh key to clone our github repos)

templating

templating is the idea of changing the input or output of a system. For instance, the Jupyter book, instead of outputting the markdown files as markdown files, displays them as HTML pages (with the contents of the markdown file).

terminal

a program that makes shell visible for us and allows for interactions with it

tree objects

type of git object in git that helps store multiple files with their hashes (similar to directories in a file system)

yml

see YAML

YAML

a file specification that stores key-value pairs. It is commonly used for configurations and settings.

zsh

zsh or z shell is built on top of the bash shell and contains new features

General Tips and Resources

Skip to main content

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 boo
programmi
Brain As of
by clicking
contents s

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

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.

GitHub Interface reference

This is an overview of the parts of GitHubt from the view on a repository page. It has links to the relevant GitHub documentation for more detail.

Top of page

The very top menu with the logo in it has GitHub level menus that are not related to the current repository.

Repository specific page

Code | **Issues** | **Pull Requests** | **Actions** | **Projects** | **Security** | **Insights** | **Settings**

This is the main view of the project

Branch menu & info, file action buttons, download options (green code button)

About has basic facts about the repo, often including a link to a documentation page

File panel

the header in this area lists who made the last commit, the message of that commit, the short hash, date of that commit and the total number of commits to the project.

Releases, Packages, and Environments are optional sections that the repo owner can toggle on and off.

Releases mark certain

failed or succeeded on that commit.

the header in this area lists who made the last commit, the message of that commit, the short hash, date of that commit and the total number of commits to the project.

If there are actions on the repo, there will be a red x or a green check to indicate that if it failed or succeeded on that commit. ^^^ file list: a table where the first column is the name, the second column is the message of the last commit to change that file (or folder) and the third column is when is how long ago/when that commit was made

README file

give easy access to that version. They are related to git tags

Packages are out of scope for this course. GitHub helps you manage distributing your code to make it easier for users.

Environments are a tool for dependency management. We will cover thigns that help you know how to use this feature indirectly, but probably will not use it directly in class. This would be eligible for a build badge.

The bottom of the right panel has information about the languages in the project

Language/Shell Specific References

- bash
- C
- Python

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

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.

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

Skip to main content

```
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.

Windows Help & Notes

CRLF Warning

This is GitBash telling you that git is helping. Windows uses two characters for a new line `CR` (cariage return) and `LF` (line feed). Classic Mac Operating system used the `CR` character. Unix-like systems (including MacOS X) use only the `LF` character. If you try to open a file on Windows that has only `LF` characters, Windows will think it's all one line. To help you, since git knows people collaborate across file systems, when you check out files from the git database (`.git/` directory) git replaces `LF` characters with `CRLF` before updating your working directory.

When working on Windows, when you make a file locally, each new line will have `CRLF` in it. If your collaborator (or server, eg GitHub) runs not a unix or linux based operating system (it almost certainly does) these extra characters will make a mess and make the system interpret your code wrong. To help you out, git will automatically, for Windows users, convert `CRLF` to `LF` when it adds your work to the index (staging area). Then when you push, it's the compatible version.

git documentation of the feature