

# About this Site

## Contents

### Syllabus

- [Basic Facts](#)
- [Tools and Resources](#)
- [Grading](#)
- [Grading Policies](#)
- [Support](#)
- [General URI Policies](#)
- [Office Hours & Comms](#)

### Activities

- [KWL Chart](#)
- [Prepare for the next class](#)
- [More Practice](#)
- [Deeper Explorations](#)
- [Project Information](#)

### FAQ

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

### Resources

- [General Tips and Resources](#)
- [How to Study in this class](#)
- [Getting Help with Programming](#)
- [Getting Organized for class](#)
- [Advice from Spring 2022 Students](#)

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

This class meets MW 4:30-5:45 in Engineering Building Room 045.

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

Course Calendar on BrightSpace

#### Tip

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

## Navigating the Sections

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

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

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

## Reading each page

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

### Try it Yourself

Notes will have exercises marked like this

### Question from Class

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

### Further reading

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

### Hint

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

### Think Ahead

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

### Ram Token Opportunity

Chances to earn ram tokens are highlighted this way.

### Question from class

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

## Basic Facts

### About this course

### About this syllabus

This syllabus is a *living* document. You can get notification of changes from GitHub by “watching” the You can view the date of changes and exactly what changes were made on the Github [commit history](#) page.

Creating an [issue](#) is also a good way to ask questions about anything in the course it will prompt additions and expand the FAQ section.

## About your instructor

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

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

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

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

## Tools and Resources

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

All of the tools and resources below are either:

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

## BrightSpace

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

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

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

### Note

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

## Prismia chat

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

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

## Course Manual

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

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

## GitHub

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

with https, you will need to [create a Personal Access Token](#) for each device you use. In order to use the command line with SSH, set up your public key.

## Programming Environment

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

### Warning

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

### Requirements:

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

### Warning

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

### Note

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

### Recommendation:

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

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

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

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

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

5. Next install Anaconda's Python from the website provided by the instructor and use the top download link under the Linux options.
6. You will then see a .sh file in your downloads, move this into your Linux files.
7. Make sure you are in your home directory (something like home/YOURUSERNAME), do this by using the `pwd` command.
8. Use the `bash` command followed by the file name of the installer you just downloaded to start the installation.

9. Next you will add Anaconda to your Linux PATH, do this by using the `vim .bashrc` command to enter the `.bashrc` file, then add the `export PATH=/home/YOURUSERNAME/anaconda3/bin/:$PATH` line. This can be placed at the end of the file.
10. Once that is inserted you may close and save the file, to do this hold escape and type `:x`, then press enter. After doing that you will be returned to the terminal where you will then type the `source .bashrc` command.
11. Next, use the `jupyter notebook --generate-config` command to generate a Jupyter Notebook.
12. Then just type `jupyter lab` and a Jupyter Notebook should open up.

Video install instructions for Anaconda:

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

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

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

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

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

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

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

## Grading

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

### Learning Outcomes

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

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

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

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

### Principles of Grading

Learning happens through practice and feedback. My goal as a teacher is for you to learn. The grading in this course is based on your learning of the material, whether it takes one try or multiple tries.

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

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

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

## No Grade Zone

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

## Grading Contract

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

Most work will be small, frequent activities, but for an A you will also do larger, more in depth activities.

All contracts will include maintaining a KWL Chart for the duration of the semester, coming to class prepared, participating in class activities, and collaborating with peers to maintain reference materials.

## Grading Policies

### Deadlines

You will get feedback on items at the next feedback period after it is submitted. During each feedback hours (twice per week) you can get feedback on new submissions from up to 2 class sessions and revision feedback on an unlimited number of submissions.

#### Important

Work does not have specific deadlines, to give you more flexibility, but to ensure timely feedback and to be fair to me at the end of the semester, there is a limit of how much material you can get feedback at a time. The 2 class session limit means that you should aim to complete things within about 1 week most of the time, but no more than 2 weeks to ensure that all of your work can be reviewed.

### Makeup Work

If you have extenuating circumstances and need to submit a large amount of work at once, first submit a PR to your grading contract outlining your plan to get caught back up for approval. Requests will typically be approved, but having a plan is required.

### Regrading

Re-request a review on your Feedback Pull request.

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

For specific questions, reply to a specific comment.

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

## Support

### Academic Enhancement Center

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

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

## General URI Policies

### Anti-Bias Statement:

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

## Disability Services for Students Statement:

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

## Academic Honesty

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

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

## URI COVID-19 Statement

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

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

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

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

## Office Hours & Comms



```

-----
FileNotFoundError                                Traceback (most recent call last)
Cell In [1], line 6
      3 pd.set_option('display.max_colwidth', 0)
      4 # df = pd.read_csv('../_data/communication.csv')
----> 6 help_df = pd.read_csv('../_data/help_hours.csv')

File /opt/hostedtoolcache/Python/3.8.13/x64/lib/python3.8/site-
packages/pandas/util/_decorators.py:311, in deprecate_nonkeyword_arguments.
<locals>.decorate.<locals>.wrapper(*args, **kwargs)
    305 if len(args) > num_allow_args:
    306     warnings.warn(
    307         msg.format(arguments=arguments),
    308         FutureWarning,
    309         stacklevel=stacklevel,
    310     )
--> 311 return func(*args, **kwargs)

File /opt/hostedtoolcache/Python/3.8.13/x64/lib/python3.8/site-
packages/pandas/io/parsers/readers.py:678, in read_csv(filepath_or_buffer, sep,
delimiter, header, names, index_col, usecols, squeeze, prefix, mangle_dupe_cols,
dtype, engine, converters, true_values, false_values, skipinitialspace, skiprows,
skipfooter, nrows, na_values, keep_default_na, na_filter, verbose,
skip_blank_lines, parse_dates, infer_datetime_format, keep_date_col, date_parser,
dayfirst, cache_dates, iterator, chunksize, compression, thousands, decimal,
lineterminator, quotechar, quoting, doublequote, escapechar, comment, encoding,
encoding_errors, dialect, error_bad_lines, warn_bad_lines, on_bad_lines,
delim_whitespace, low_memory, memory_map, float_precision, storage_options)
    663 kwds_defaults = _refine_defaults_read(
    664     dialect,
    665     delimiter,
    (...)
    674     defaults={"delimiter": ",",
    675 )
    676 kwds.update(kwds_defaults)
--> 678 return _read(filepath_or_buffer, kwds)

File /opt/hostedtoolcache/Python/3.8.13/x64/lib/python3.8/site-
packages/pandas/io/parsers/readers.py:575, in _read(filepath_or_buffer, kwds)
    572 _validate_names(kwds.get("names", None))
    574 # Create the parser.
--> 575 parser = TextFileReader(filepath_or_buffer, **kwds)
    577 if chunksize or iterator:
    578     return parser

File /opt/hostedtoolcache/Python/3.8.13/x64/lib/python3.8/site-
packages/pandas/io/parsers/readers.py:932, in TextFileReader.__init__(self, f,
engine, **kwds)
    929 self.options["has_index_names"] = kwds["has_index_names"]
    931 self.handles: IOHandles | None = None
--> 932 self._engine = self._make_engine(f, self.engine)

File /opt/hostedtoolcache/Python/3.8.13/x64/lib/python3.8/site-
packages/pandas/io/parsers/readers.py:1216, in TextFileReader._make_engine(self,
f, engine)
    1212 mode = "rb"
    1213 # error: No overload variant of "get_handle" matches argument types
    1214 # "Union[str, PathLike[str], ReadCsvBuffer[bytes], ReadCsvBuffer[str]]"
    1215 #, "str", "bool", "Any", "Any", "Any", "Any", "Any", "Any"
-> 1216 self.handles = get_handle( # type: ignore[call-overload]
    1217     f,
    1218     mode,
    1219     encoding=self.options.get("encoding", None),
    1220     compression=self.options.get("compression", None),
    1221     memory_map=self.options.get("memory_map", False),
    1222     is_text=is_text,
    1223     errors=self.options.get("encoding_errors", "strict"),
    1224     storage_options=self.options.get("storage_options", None),
    1225 )
    1226 assert self.handles is not None
    1227 f = self.handles.handle

File /opt/hostedtoolcache/Python/3.8.13/x64/lib/python3.8/site-
packages/pandas/io/common.py:786, in get_handle(path_or_buf, mode, encoding,
compression, memory_map, is_text, errors, storage_options)
    781 elif isinstance(handle, str):
    782     # Check whether the filename is to be opened in binary mode.
    783     # Binary mode does not support 'encoding' and 'newline'.
    784     if ioargs.encoding and "b" not in ioargs.mode:
    785         # Encoding
--> 786         handle = open(
    787             handle,
    788             ioargs.mode,
    789             encoding=ioargs.encoding,
    790             errors=errors,
    791             newline="",
    792 )

```

```
793     else:
794         # Binary mode
795         handle = open(handle, ioargs.mode)

FileNotFoundError: [Errno 2] No such file or directory: '../_data/help_hours.csv'
```

## Help Hours

TBA


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

## Tips

### For assignment help

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

### Using issues

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

### For E-mail

- use e-mail for general inquiries or notifications
- Please include `[csc392]` in the subject line of your email along with the topic of your message. This is important, because your messages are important, but I also get a lot of e-mail. Consider these a cheat code to my inbox: I have setup a filter that will flag your e-mail if you include that in subject to ensure that I see it.

## 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-specific branch to `main`.
3. when it is approved, merge into main.

### Minimum Rows

#### Tip

You could apply branch protections on your feedback branch if you like

## # KWL Chart

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

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

## Prepare for the next class

### Warning

these are listed by the date they were *posted* (eg the content here under Feb 1, was posted Feb 1, and should be done before the Feb 3 class)

*below* refers to following in the notes

```
import os
from IPython.display import Markdown, display

prep_file_list = sorted(os.listdir('../_prepare/'))
```

```
-----
FileNotFoundError                                Traceback (most recent call last)
Cell In [1], line 4
      1 import os
      2 from IPython.display import Markdown, display
----> 4 prep_file_list = sorted(os.listdir('../_prepare/'))

FileNotFoundError: [Errno 2] No such file or directory: '../_prepare/'
```

```
for prep_file in prep_file_list:
    date_str = prep_file[:-3]
    date_link = '[' + date_str + '](..notes/' + date_str + ']'
    display(Markdown(date_link))
    display(Markdown('../_prepare/' + prep_file))
```

```

-----
NameError                                Traceback (most recent call last)
Cell In [2], line 1
----> 1 for prep_file in prep_file_list:
      2     date_str = prep_file[:-3]
      3     date_link = '[' + date_str + '](..notes/' + date_str + ')'

NameError: name 'prep_file_list' is not defined

```

## More Practice

### Note

these are listed by the date they were *posted*

```

import os
from IPython.display import Markdown, display

prep_file_list = sorted(os.listdir('../_practice/'))

```

```

-----
FileNotFoundError                        Traceback (most recent call last)
Cell In [1], line 4
      1 import os
      2 from IPython.display import Markdown, display
----> 4 prep_file_list = sorted(os.listdir('../_practice/'))

FileNotFoundError: [Errno 2] No such file or directory: '../_practice/'

```

```

for prep_file in prep_file_list:
    date_str = prep_file[:-3]
    date_link = '[' + date_str + '](..notes/' + date_str + ')'
    display(Markdown(date_link))
    display(Markdown('../_practice/' + prep_file))

```

```

-----
NameError                                Traceback (most recent call last)
Cell In [2], line 1
----> 1 for prep_file in prep_file_list:
      2     date_str = prep_file[:-3]
      3     date_link = '[' + date_str + '](..notes/' + date_str + ')'

NameError: name 'prep_file_list' is not defined

```

## Deeper Explorations

### Warning

deeper explorations are not required, but an option for higher grades

If your contract includes that you will complete deeper explorations, this page includes guidance for what is expected.

Deeper explorations can take different forms so the sections below outline some options, it is not a cumulative list of requirements.

### 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.

### How to get it reviewed?

Follow the workflows for your [kwl repo](#) and tag the instructors for a review.

## What should the work look like?

It should look like a blog post or written tutorial. It will likely contain some code excerpts the way the 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.

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

## Project Information

### Proposal Template

If you have selected to do a project, please use the following template to add a section to the end of your `contract.md`

```
## < Project Title >

<!-- 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 solve. this should be 2-5 sentences, it can be bullet
points/numbered or a paragraph -->

### method

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

### deliverables

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

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 summarized
- if you are researching something, a report should be 2-4 pages 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

These can be whatever form is appropriate to your specific project. Your proposal should indicate what form those will take.

## Summary Report

This summary report will be added to the grading contract repo as a new file `project_report_title.md` where title is the title from the project 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.

## 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 fulfilling your contract. If all items are completed to a satisfactory level, then you earn that grade, if not, then you will be prompted to revise the contract to signal that you are aware you have completed fewer items.

### I don't understand the feedback on this assignment

If you have questions about your grade, the best place to get feedback is to reply on the Feedback PR. Either reply directly to one of the inline comments, or the summary.

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

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

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

[an example that uses mostly screenshots](#)

[an example of heavily annotated code](#)

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

## Git and GitHub

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

If your error looks like this...

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

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

After you run

```
git pull
```

You'll probably have to [resolve a merge conflict](#)

### My command line says I cannot use a password

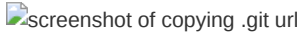
GitHub has [strong rules](#) about authentication You need to use SSH with a public/private key; HTTPS with a [Personal Access Token](#) or use the [GitHub CLI auth](#)

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

That's ok. You can fix it.

You'll have to work offline and use GitHub in your browser together for this fix. The following instructions 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
```

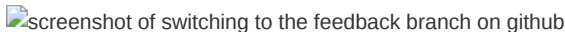
then past your url that you copied. It will look something like this, but the last part will be the current assignment repo and your username.

```
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
```

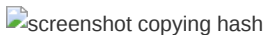
Now, you have to figure out what commit to revert to, so go back to GitHub in your browser, and switch to the feedback branch there. Click on where it says **main** on the top right next to the branch icon and choose feedback from the list.



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

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

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

Now to make GitHub match your reverted local copy.

```
git push origin -f
```

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

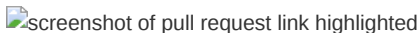
```
Total 0 (delta 0), reused 0 (delta 0)
To https://github.com/rhodyprog4ds/portfolio-brownsarahm.git
+ f301d90...822cfe5 feedback -> feedback (forced update)
```

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

Now back on GitHub, in your browser, click on the code tab. It should look something like this now. Notice that it says, "This branch is 11 commits behind main" your number will be different but it should be 1 less than the number you had when you checked `git status`. This is because we reverted the changes you made to `main` (11 for me) and the 1 commit for merging `main` into `feedback`. Also the last commit (at the top, should say "Setting up GitHub Classroom Feedback").

screenshot of feedback branch code tab after revert

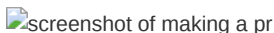
Now, you need to recreate your Pull Request, click where it says pull request.

screenshot of pull request link highlighted

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

screenshot of changing pr base to feedback

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

screenshot of making a pr

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

Now you're done!

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



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

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

## General Tips and Resources

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

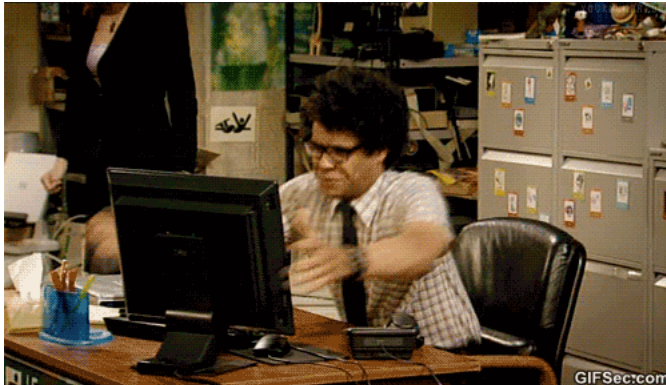
### on email

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

## How to Study in this class

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

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



### Why this way?

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

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

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

## Learning in class

### ! Important

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

Programming requires both practical skills and abstract concepts. During class time, we will cover the practical aspects and introduce the basic concepts. You will get to see the basic practical details and real examples of debugging during class sessions. Learning to debug something you've never encountered before and setting up your programming environment, for example, are *high frustration*

activities, when you're learning, because you don't know what you don't know. On the other hand, diving deeper into options and more complex applications of what you have already seen in class, while challenging, is something I'm confident that you can all be successful at with minimal frustration once you've seen basic ideas in class. My goal is that you can repeat the patterns and processes we use in class outside of class to complete assignments, while acknowledging that you will definitely have to look things up and read documentation outside of class.

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

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

## After class

After class, you should practice with the concepts introduced.

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


When you review the notes, you should be adding comments on tricky aspects of the code and narrative text between code blocks in markdown cells. While you review your notes and the annotated course notes, you should also read the documentation for new modules, libraries, or functions introduced in that class. We will collaboratively annotate notes for this course. Dr. Brown will post a basic outline of what was covered in class and we will all fill in explanations, tips, and challenge questions. Responsibility for the main annotation will rotate.

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

## Getting Help with Programming

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

## Asking Questions

 comic on asking questions, that summarizes blog post

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

## Describing what you have so far

Stackoverflow is a common place for programmers to post and answer questions.

As such, they have written a good [guide on creating a minimal, reproducible example](#).

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

### Note

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

## Getting Organized for class

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

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

## File structure

I recommend the following organization structure for the course:

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

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

Please **do not** include all of your notes or your other assignments all inside your portfolio, 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 account. Since your account is not the owner, they do not show on your profile.

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

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

### Warning

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