



Lab 01: Setup

FAQ

Introduction

[Goals and Outcomes](#)

[Welcome to CS 61B!](#)

[Partners](#)

Personal Computer Setup

[Installing Software](#)

The Terminal

[How to use the Terminal](#)

[Terminal Test Run](#)

GitHub and Beacon

[Account Setup](#)

[Your Repository](#)

Git

[Git Basics](#)

[Setting Up Git](#)

[Git Exercise](#)

[Git and Remote Repos](#)

[Setting up your Git Repository](#)

[Getting the Skeleton](#)

[Pushing to GitHub](#)

Submitting to Gradescope

Setting Up Java Libraries

IntelliJ Setup

[Installing IntelliJ](#)

[Installing Plugins](#)

[Creating Projects](#)[IntelliJ Test](#)[Testing Your Code](#)[Saving Your Work using Git and GitHub](#)[Deliverables](#)[Optional: Josh Hug's Color Scheme IntelliJ Download](#)

FAQ

Each assignment will have an FAQ linked at the top. You can also access it by adding “/faq” to the end of the URL. The FAQ for Lab 1 is located [here](#).

Introduction

In this lab, we will prepare ourselves to succeed in the course by setting up our computers, so that we can use them to work on assignments. We will also write a small Java program to learn the assignment workflow.

Goals and Outcomes

In this lab, you will set up your computer for the class. You will also read and understand a short Java program, and identify its logical flaws. By the end of this lab, you will:

- Have a computer that you can use to work on class assignments.
 - Be able to get the starter code for an assignment.
 - Have written and submitted a small Java program.
-

Welcome to CS 61B!

We're very excited to work with you this term! Before we can get started, you'll need to have a computer that you can work on assignments on. In this class, you'll be using real-world tools, which means that you'll likely run into real-world problems. These are hard problems, and software engineers run into them every day! **Don't be discouraged**, and make sure to ask for help if you're stuck!

We highly recommend attending a lab section, and asking for help during lab section. If you're working outside of a lab section, you can ask on Ed or attend office hours.

⚠ If something isn't working, **do not** keep trying random things! This will make it **much** more difficult for us to identify and fix the problem later.

Instead, ask for help. Your lab TA will tell you how to join the queue. They may choose to use a whiteboard queue or the [online OH queue](#).

In general, while you are waiting, you should **move on to the next step of the assignment**, whenever possible.

Partners

Labs in CS 61B are **solo**. This means that you will need to write and submit your own code. For Lab 1 in particular, you will need to set up your own computer.

However, we strongly encourage working together with other students! In-person lab sections are an excellent place to find groups.

Personal Computer Setup

Installing Software

Installation will vary, depending on your operating system.

- [Windows instructions](#)

- [macOS instructions](#)
- [Linux instructions](#)

 **Task:** Follow the guide for your operating system to install the software.

The Terminal

How to use the Terminal

In CS 61B, we will be using the terminal extensively to work with git. The terminal also has some other commands that allow you to work with folders or files. Here's a list of the most important commands for CS 61B:

- `cd`: change your working directory

```
cd hw
```

This command will change your directory to `hw`.

- `pwd`: present working directory

```
pwd
```

This command will tell you the full absolute path for the current directory you are in if you are not sure where you are.

- `~`: shorthand for your home directory
- `.`: shorthand for your current directory

```
cd .
```

This command will change your directory to the current directory (aka. do nothing).

- `..`: shorthand for one parent directory above your current directory

```
cd ..
```

This command will change your directory to its parent. If you are in `/workspace/day1/`, the command will place you in `/workspace/`.

- `ls`: list files/folders in a directory

```
ls
```

This command will list all the files and folders in your current directory. You can also use `ls <directory>` to list the contents of a different directory – try `ls ..`!

- `mkdir`: make a directory

```
mkdir [dirname]
```

This command will make a new directory within the current directory called `dirname`.

- `touch`: create a new file

```
touch [filename]
```

This command will create a file within the current directory called `filename`.

- `rm`: remove (delete) a file

```
rm [file]
```

This command will remove `file` from the current directory. It will not work if `file` does not exist.

```
rm -r [dir]
```

This command will remove the `dir` directory recursively. In other words, it will delete all the files and directories in `dir` in addition to `dir` itself. Be careful with this command!

- `cat`: display the contents of a file

```
cat [file]
```

This command is useful for inspecting the contents of files in the terminal without having to open them in a program. It is not as useful for large files (which can clutter the terminal) or non-text files (which will likely output gibberish).

- `cp`: copy a file

```
cp lab1/original lab2/duplicate
```

This command will copy the `original` file in the `lab1` directory and create a `duplicate` file in the `lab2` directory.

- `mv`: move or rename a file

```
mv lab1/original lab2/original
```

This command moves `original` from `lab1` to `lab2`. Unlike `cp`, `mv` does not leave original in the `lab1` directory.

```
mv lab1/original lab1/newname
```

This command does not move the file but rather renames it from `original` to `newname`.

Here are some other useful tricks when working in a terminal.

- Your shell can complete file names and directory names for you with *tab completion*. When you have an incomplete name (for something that already exists), try pressing the `tab` key for autocomplete or a list of possible names.
- You can copy-paste into the terminal. This is straightforward on Mac, but on Windows, right-click to copy and paste highlighted text.
- If you want to run the same command used recently, press the `up` arrow key on your keyboard until you see the correct instruction. If you

go too far, use the `down` key to go back. This saves typing time if you are doing repetitive instructions.

Terminal Test Run

Let's ensure that everything is working.

1. First open up your terminal. Check that git is a recognized command by typing the following command:

```
git --version
```

The version number for git should be printed. If you see “git: command not found”, or similar, try opening a new terminal window, restarting your computer, or installing git again.

2. Second, let's check that `javac` and `java` are working. `javac` and `java` allow *Command Line Compilation*, or in other words, the ability to run Java programs directly from the command line. In practice, most developers run Java programs through an IDE like IntelliJ, so we won't be using command line compilation this semester other than testing your setup.

Start by running the following commands at your terminal.

```
mkdir ~/temp  
cd ~/temp
```

1. Then, open your operating system's file explorer in this directory. You can do this from the command line:

- Mac: `open .`
- Windows: `explorer .`
- Ubuntu: `gnome-open .`

2. In this newly opened directory, create a file `HelloWorld.java` with these contents:

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello world!");  
    }  
}
```

Note that in addition to the file finder or explorer, you are able to create a file using the `touch` in the terminal; try creating the file using `touch HelloWorld.java` from the terminal! Then, open the file using your favorite text editor and copy-paste the code.

3. In your terminal, enter `ls` (list the files/folders in this directory). You should see `HelloWorld.java` listed.
4. Run `javac HelloWorld.java`. If this produces any output, then something may be wrong with your setup. Try opening a new terminal window or restarting your computer. If that still doesn't work, see the Troubleshooting section under the directions for your operating system.
5. Type `ls`, you should see both `HelloWorld.java` and a freshly created `HelloWorld.class` (the `javac` command created this file).
6. Run `java HelloWorld`. It should print out "Hello world!" for you. If it didn't, something is wrong with your setup!
7. You're done! You can also delete the "temp" folder and its contents as you please.

The screenshot below shows what we're hoping for when we do the above steps. If you see something similar to this, your terminal setup is complete.

GitHub and Beacon


Instead of bCourses, CS 61B uses an in-house system for centralizing your grades and student information called Beacon.

In this section, we'll set up your Beacon account as well as your CS 61B GitHub repository ("repo"), which you will need to submit all coding assignments.

Account Setup

1. Create an account on [GitHub](#). If you already have an account, you do not need to create a new one.
2. Go to [Beacon](#) and follow the steps to complete your GitHub repository registration. You must be logged in to your Berkeley account to complete the Google Form syllabus quiz.
3. After completing all of the steps, you should receive an email inviting you to collaborate on your course GitHub repository. Accept the email invitation to receive access to your course repository. This email will be sent to the **email that you used to create your GitHub account, which may not necessarily be your Berkeley email.**

⚠ Don't follow the instructions that GitHub says you might want to do. We have our own set of instructions later in this lab.

 **Task:** Follow the steps above to create your GitHub and Beacon accounts, and connect them.

Your Repository

Your repository will have a name containing a number that is unique to you. For instance, if your repo is called "`sp23-s1`", you'll be able to visit your private repository at <https://github.com/Berkeley-CS61B-Student/sp23-s1> (when logged into GitHub). Your student number is not "1", so this link will

not work for you. Replace “1” with your own number to see your repo on GitHub.

Additionally, the instructors, TAs, and tutors will be able to view your repository. This means you can (and should!) link to your code when asking private debugging questions in Ed or Gitbugs. No other students will be able to view your repository.

⚠ As a reminder, you may not post code from this course publicly, even after completing the course. Doing so is a violation of our course policies and you might be subject to disciplinary action.

Git

Git Basics

In this course, you'll be required to use the Git version control system, which is nearly universal out in the real world. Since the abstractions behind it are fairly tricky to understand, don't be worried if you encounter significant frustration as you learn to use it.

✍ **Task:** Before you proceed, **read up to the Remote Repositories section of the Using Git Guide**. You do not need to read past that.

Setting Up Git

Before we use git, we have some short commands to configure it appropriately.

First, set the name and email that git will use with these two commands:

```
git config --global user.name "<your name>"
git config --global user.email "<your email>"
```

Set git's default branch name:


```
git config --global init.defaultBranch main
```

Set the “merge strategy”:

```
git config --global pull.rebase false
```

We'll also change the text editor associated with git. Sometimes, git needs your help when inputting things like commit messages, so it will open a text editor for you. The default editor is `vim`, which is notoriously difficult to use. We recommend `nano` for this course, but you're free to use whatever you'd like.

Follow the instructions [here](#). This will configure Git's default editor (make sure that you follow the correct instructions for your operating system).

 **Task:** Configure git by following the above instructions, and set your preferred editor.

Git Exercise

Now you're ready to start using git! Your next task is to work through a small git workflow by setting up a repository and making a couple commits. At the end, you will need to be checked off by filling out the form linked on Beacon.

If you need help with creating directories, creating files, changing directories, etc., refer back to [How to use the Terminal](#).

[Section C of the Using Git Guide](#) will also be helpful for this exercise.

1. Create a directory called `lab01-checkoff`. You can put this directory anywhere on your computer (unless you have already cloned your `sp23-s***` repository, in which case, you **should not put this directory inside of your `sp23-s***` repo**).
2. Move into the `lab01-checkoff` directory, and initialize a git repository in this directory.
3. Create a file called `61b.txt` in any way you'd like. In this text file, add the text “61b version 1” into it.

4. Create another file called `61bee.txt` in any way you'd like. file, add the text "61bee version 1" into it.
5. Begin tracking **only** `61b.txt`, and create a new commit containing just this file, with the following commit message: `Add 61b.txt`.
6. Make a modification in `61b.txt` by changing the text in the file to: "61b changed to version 2".
7. Make another commit, this time containing both `61b.txt` and `61bee.txt`. The commit message should be: `Update 61b.txt and add 61bee.txt`.
8. Make one more modification to `61b.txt` by changing the text in the file to: "61b changed to version 3". **Don't commit this version.**

At this point, if you were to type in `git status` and `git log`, you'd see something like the following:

9. **Using git only**, restore `61b.txt` to the version in the most recent commit.
10. **Using git only**, restore `61b.txt` to the version in the first commit.

Be sure to save this repository and directory until you complete the [asynchronous checkoff form on beacon](#) and obtain a **magic word**.

 **Task:** Do the steps above, then get checked off by filling out the Beacon form.

Git and Remote Repos

First, read the **Remote Repositories** section of the [Using Git Guide](#).

In this course, you'll be required to submit your code using git to your course GitHub repository that you created in [Account Setup](#). This is for several reasons:

- To spare you the incredible agony of losing your files.
- To submit your work for grading and to get results back from the autograder.
- To save you from the tremendous anguish of making unknown changes to your files that break everything.
- To ensure that we have easy access to your code so that we can help if you're stuck.
- **To dissuade you from posting your solutions on the web in a public GitHub repository.** This is a major violation of course policy!
- To expose you to a realistic workflow that is common on every major project you'll ever work on in the future.

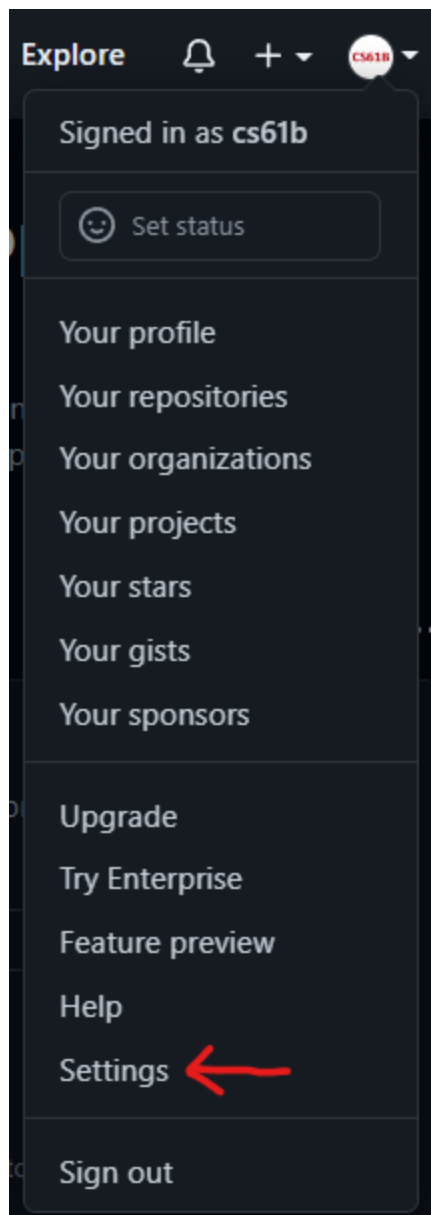
Setting up your Git Repository

Clone your sp23-s*** Git Repository

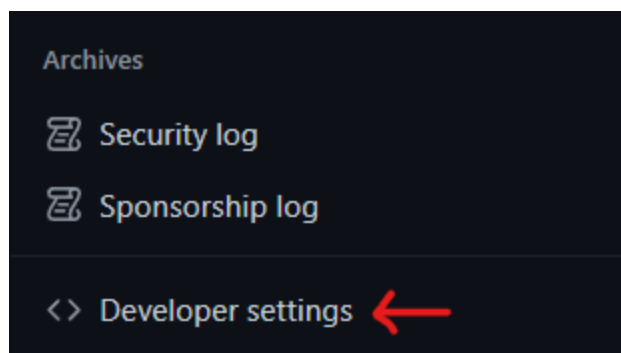
Navigate to the spot in your folders on your computer that you'd like to start your repository. In the example below, we're assuming you want all your stuff in a folder named `cs61b`, but you can pick a different name if you'd like.

```
cd cs61b
```

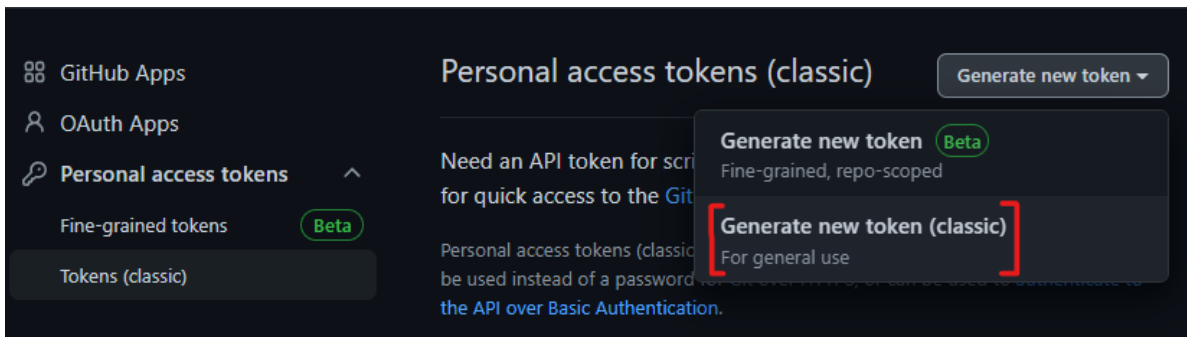
In order to clone your GitHub repo, your local computer needs to setup GitHub access credentials. Go to `github.com`, sign in with your GitHub username and password, and click the user profile in the top right corner, then click to "Settings":



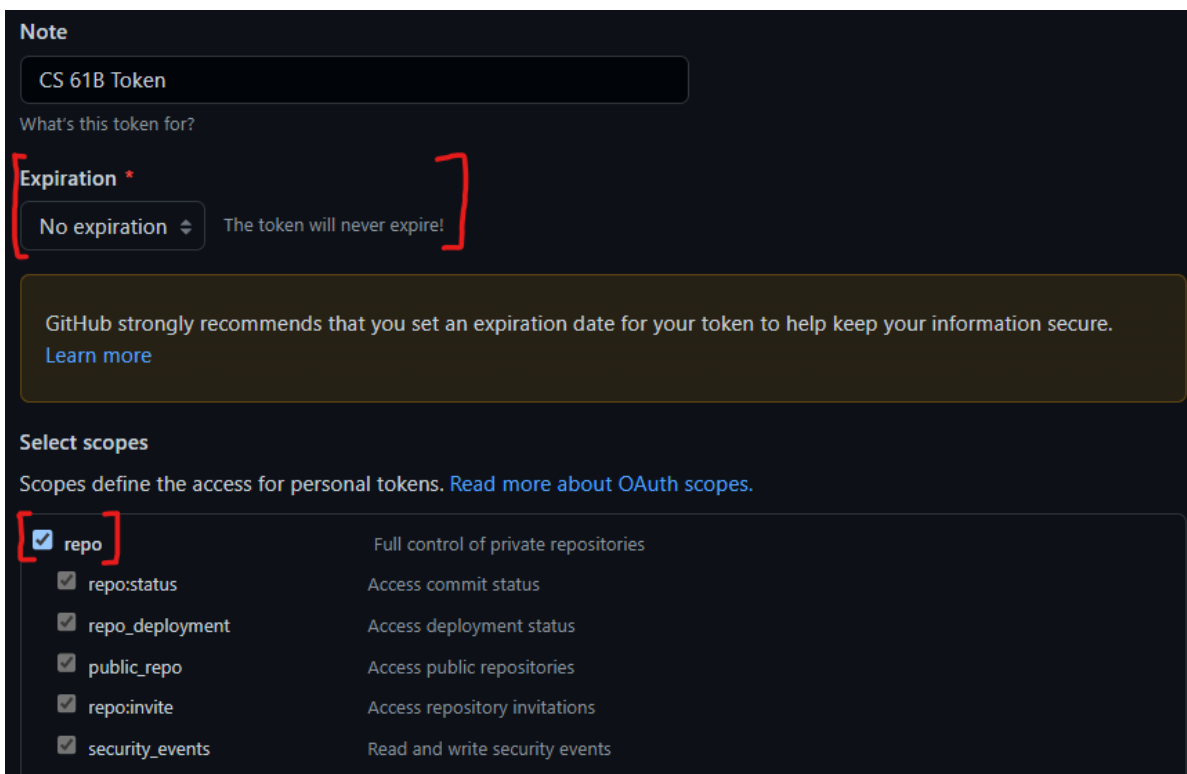
Then click on the “developer settings” tab:



and generate a **classic** personal access token.



- Give a note to the token so you remember what it's for
- Set the token to have no expiration
- Give the token “repo” permissions



Then, when you're ready, scroll to the bottom of the page and press the green “Generate Token” button.

⚠ Be sure to copy the token GitHub presents. This token is a password; do not upload it to the internet or give it to anyone. Someone can use this token to delete your GitHub repos.

After obtaining a token, enter the following command to clone your GitHub repo. Make sure to replace the `***` with your repository number from Beacon.

```
git clone https://github.com/Berkeley-CS61B-Student/sp23-s***.git
```

You might be prompted for a username and password:

```
[Todds-MacBook-Pro:temp toddyu$ git clone https://github.com/Berkeley-CS61B/websi]
te-fa22.git
Cloning into 'website-fa22'...
Username for 'https://github.com': todd-yu
Password for 'https://todd-yu@github.com':
```

Enter your GitHub username. For the password, use the token you generated. When you copy-paste, it won't show anything – this is expected.

i After cloning your terminal will report

warning: You appear to have cloned an empty repository. This is not an issue, it is just git letting you know that there are no files in the repo.

You might instead be prompted to sign in in a browser window for “Git Credential Manager,” or not be prompted to sign in at all. As long as you're able to clone the repo successfully, the exact sign in method does not matter.

Move into your newly created repo!

```
cd sp23-s***
```

Make sure that we're working with the branch name we expect, `main`:

```
git branch -M main
```

Add the `skeleton` remote repository. We add starter code for the assignments to `skeleton`, and you will pull from it.

```
git remote add skeleton https://github.com/Berkeley-CS61B/skeleton-
```




Listing the remotes should now show both the `origin` and `skeleton` remotes.

```
git remote -v
```

i If you see an error like `fatal: not a git repository` make sure you have properly moved into the `sp23-s***` directory using `cd`.

 **Task:** Follow the steps above to clone and configure your repository.

Getting the Skeleton

 **Task:** Follow the instructions in the [Getting the Skeleton section of the Assignment Workflow guide](#) to get the skeleton code for Lab 1.

⚠ At this point, you should have a `lab01` folder, with the contents `src/Arithmetic.java` and `tests/ArithmeticTests.java`. If you **do not have these contents**, don't make it manually! Instead, pull from the skeleton or ask a staff member.

Pushing to GitHub

! You will need the magic word (obtained from checkoff) to complete this step.

Open the file `lab01/magic_word.txt` in a text editor, and edit it to contain the **magic word** obtained during the [git exercise](#).

Now stage and commit `magic_word.txt` (make sure you're in your repo!).


```
git add lab01/magic_word.txt
git commit -m "Added Magic Word"
```

Right now, the modified `magic_word.txt` is only on your computer. We want to push these changes to the GitHub repository so that your changes

can be seen by us and Gradescope. Push these changes to the `main` branch on the `origin` remote repo.

```
git push origin main
```


You can verify that this was successful by checking your repository online on GitHub's website. It should contain the updated `magic_word.txt` file. If it doesn't, make sure that your `add` and `commit` were successful. In particular, make sure that you are in your repo, `sp23-***`.

 **Task:** Follow the instructions above to push your magic word to GitHub, and check that it appears.


Our work is now on GitHub, and ready to submit!

Submitting to Gradescope

Although we use GitHub to store our programming work, we use **Gradescope** to actually grade it. The last step is to submit your work with [Gradescope](#), which we use to autograde programming assignments.

 We added everyone's CalCentral email to Gradescope on the first day of labs. Make sure to login using the email address listed on CalCentral.

If you're having trouble accessing the course on Gradescope or would like to use a different email address, ask your TA!

 As above, **we strongly encourage you to make frequent commits!** Lack of proper version control will not be considered an excuse for lost work, particularly after the first week.

 **Task:** Follow the instructions in the [Submitting to Gradescope section of the Assignment Workflow guide](#) to submit to Gradescope.

► Once you submit, open this dropdown. No, you're not done with the lab yet.

Setting Up Java Libraries

Like in Python, we sometimes want to use libraries that others wrote. Java dependency management is a bit of a mess, so we instead provide a git repo that contains all the dependencies that we will use in this course.

⚠ **First, move out of your `sp23-s***` repo with `cd ..`. Failing to do so can cause many headaches later.**

Then, run:

```
git clone https://github.com/Berkeley-CS61B/library-sp23
```

Below is shown the directory structure of `library-sp23`. Look inside the folder using `ls library-sp23` and make sure you see the `.jar` files listed below. There are many more, but we only list the first few. If you're using your operating system's file explorer, the `jar` part might not show up in the filenames, and that's OK.

```
library-sp23
├── algs4.jar
├── animated-gif-lib-1.4.jar
├── antlr4-runtime-4.11.1.jar
├── apiguardian-api-1.1.2.jar
└── ...
```

✍ **Task:** Follow the instructions above to get the course libraries.

IntelliJ Setup

IntelliJ is an Integrated Development Environment or IDE. An IDE is a single program which combines typically a source code editor, tools to compile and run code, and a debugger. Some IDEs like IntelliJ contain even more features such as an integrated terminal and a graphical interface for git

commands. Finally, IDEs also have tools like code completion which will help you write Java faster.

We *highly* recommend using IntelliJ. Our tests are written to run in IntelliJ, and we will explicitly use its debugger in later labs. Additionally, IntelliJ is an industry-standard tool that you will almost certainly encounter if you work with Java again in the future.

We will assume that you are using IntelliJ, and will not offer support for other editors, including VSCode.

⚠️ IntelliJ is a real world, industrial software development application. There are many features that we will not use, and you will sometimes encounter situations that do not make sense. **Ask for help if you are stuck or something seems broken!** It can be very hard to guess the right thing to do in IntelliJ. Check out the [IntelliJ WTFs Guide](#) for solutions to some common problems.

Before continuing, make sure that you have completed all above tasks besides the git exercise:

1. You have installed Java 17 or higher.
2. You have successfully created your local repo for the class on your own machine. This is the `sp23-s***` repository you earlier.
3. You have pulled from the skeleton, and you have a `lab01` directory.

Installing IntelliJ

1. Download the Community Edition of IntelliJ from the [JetBrains](#) website. As a student you can actually get a student license for the Ultimate version, but there are not any additional features that we will use for this class. **It is recommended and assumed that you proceed with the Community Edition.**

💡 If you have an M1 or M2 Mac, select “.dmg (Apple Silicon)”. Otherwise, select “.dmg (Intel).”

2. After selecting the appropriate version for your OS, click download and wait a few minutes for the file to finish downloading.

3. Run the installer. If you have an older version of IntelliJ, you should uninstall it at this time and replace it with this newer version.

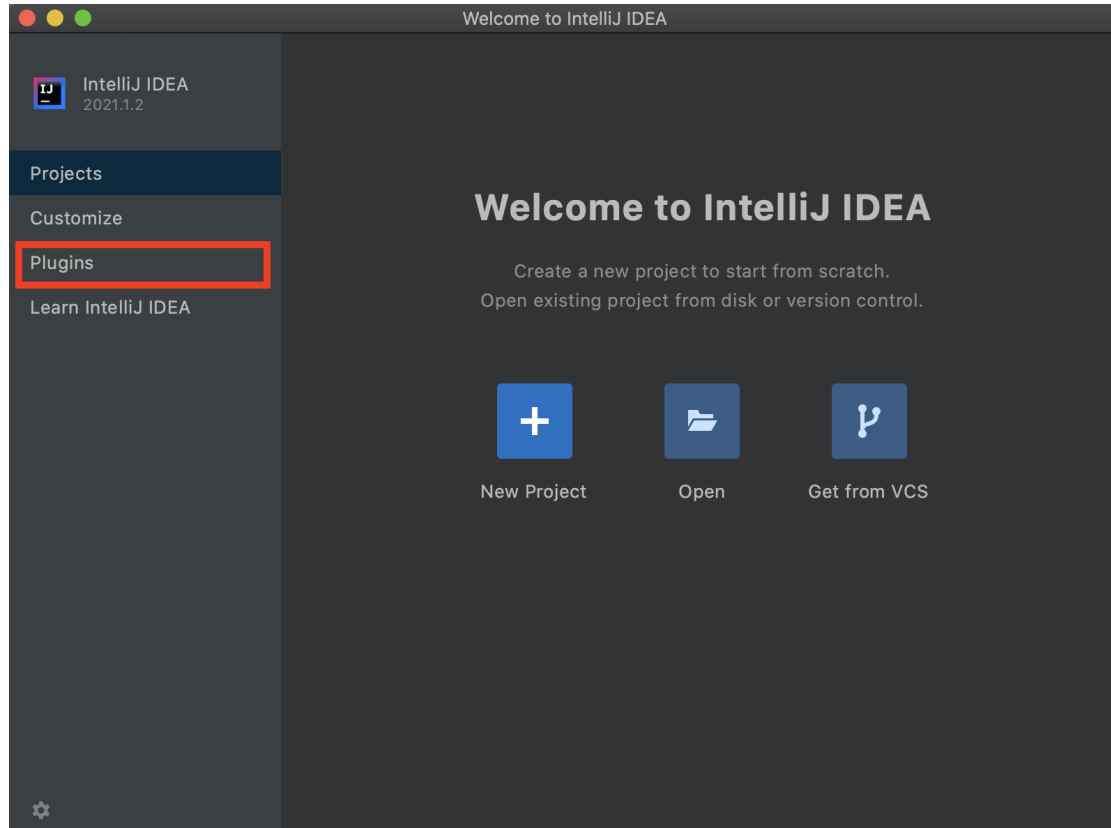
i While IntelliJ downloads, you can read / skim our [Using IntelliJ Guide](#). You don't need to read or internalize all of this to complete the lab. IntelliJ is complicated, but the core features should feel somewhat familiar to text editors you have used in the past.

Installing Plugins

Open IntelliJ. Then follow the steps below.

Make sure you're running IntelliJ Version 2021.2 or later before continuing. This is because we will use Java 17. We are using **IntelliJ Version 2022.3**. Older versions may also work but we haven't tried them ourselves.

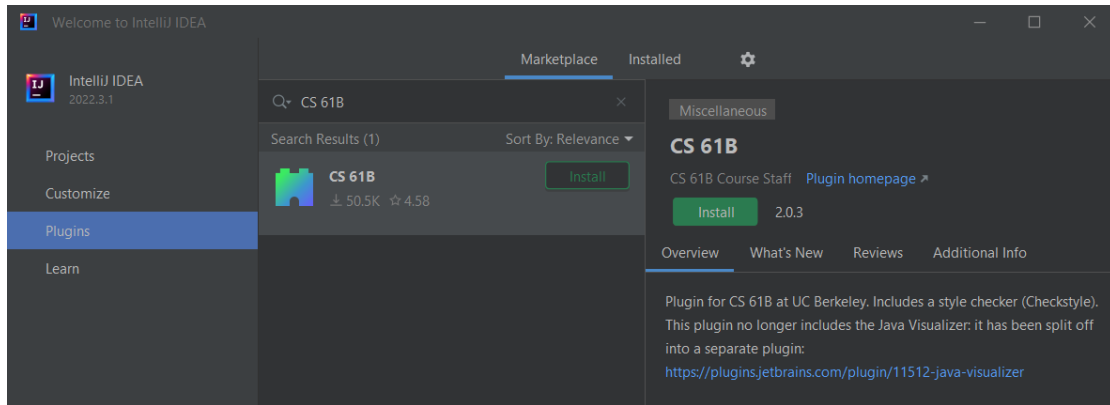
1. In the *Welcome* window, click the “**Plugins**” button in the menu on the left.



2. On the window that appears, click “Marketplace” and enter “CS 61B” in the search bar at the top. The CS 61B plugin entry should appear. If

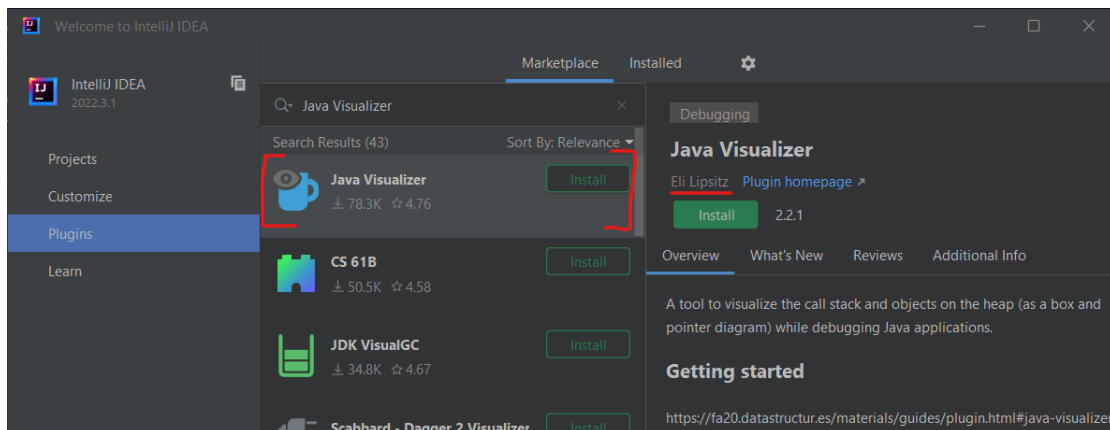
you click the autocomplete suggestion, a slightly different window from what is shown below may appear – this is okay.

3. Click the green **Install** button, and wait for the plugin to download and install.



If you have the plugin installed from a prior term, make sure to update it.


4. Now, search for “Java Visualizer”, and click the green **Install** button to install the plugin.



5. Restart (close and reopen) IntelliJ.

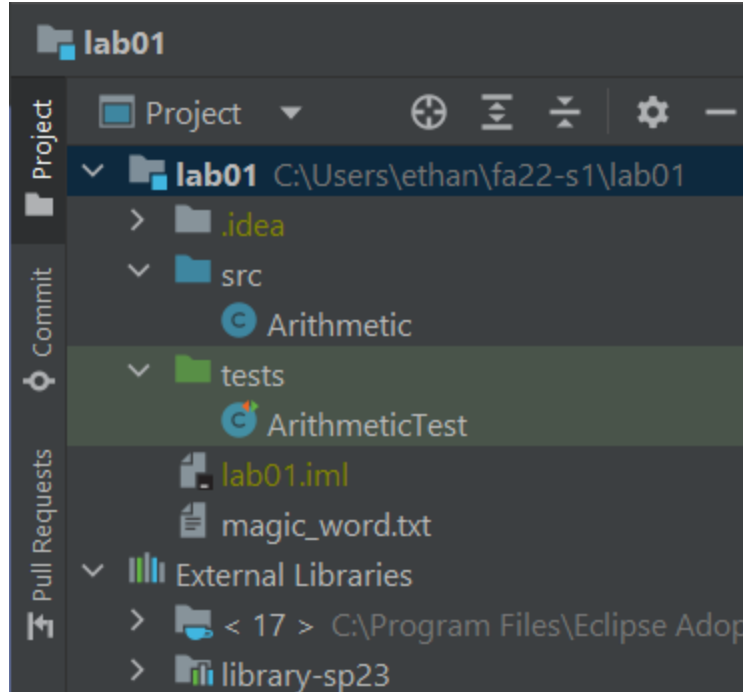
For more information on using the plugins, read [the plugin guide](#). You don't have to read this right now.

Creating Projects

 **Task:** Follow the instructions in the [Opening in IntelliJ section of the Assignment Workflow guide](#) to open `lab01`.

Once you've done this, you should see at least these three files in the left pane:

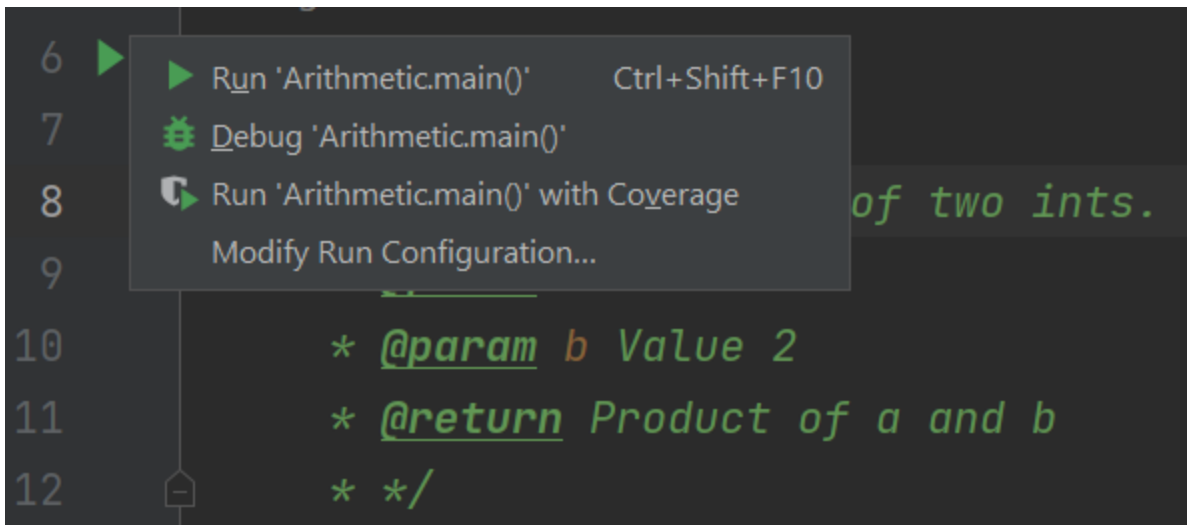
- `magic_word.txt`, which should contain the magic word you added.
- `src/Arithmetic`, a Java file which contains your first programming exercise.
- `tests/ArithmeticTest`, another Java file which will check that `Arithmetic` is implemented correctly.



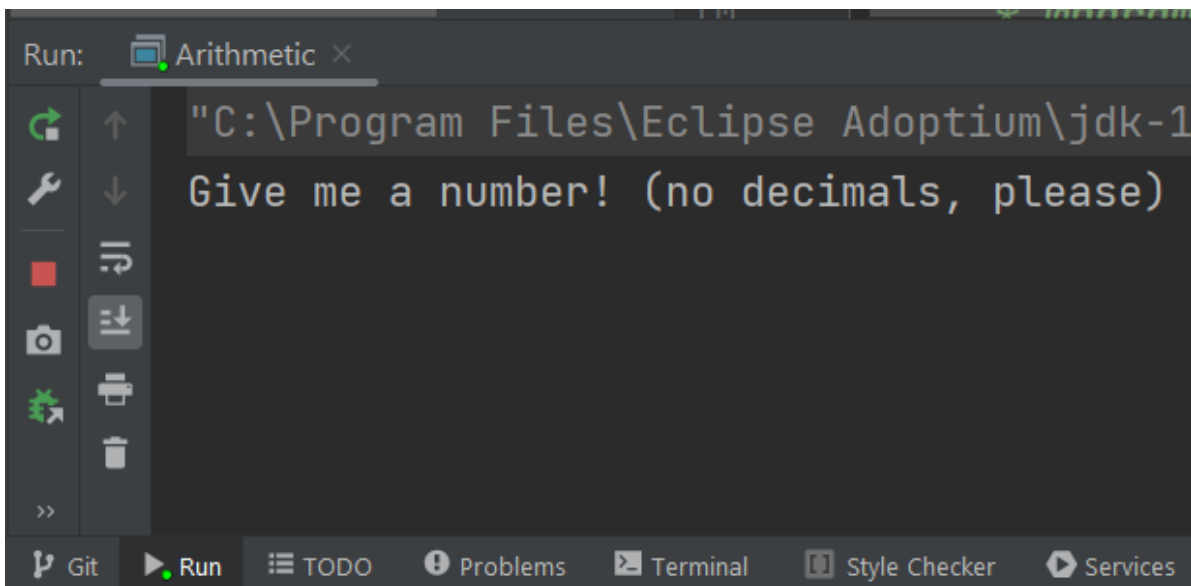
When you open `Arithmetic` and `ArithmeticTest`, you should not see any red text or red squiggles.

IntelliJ Test

To test if everything is working correctly, run the `Arithmetic` class by opening the file, clicking on the green triangle next to `public class Arithmetic`, then clicking “Run ‘Arithmetic.main()’”.



You should see a console pop up, prompting you to enter a number:

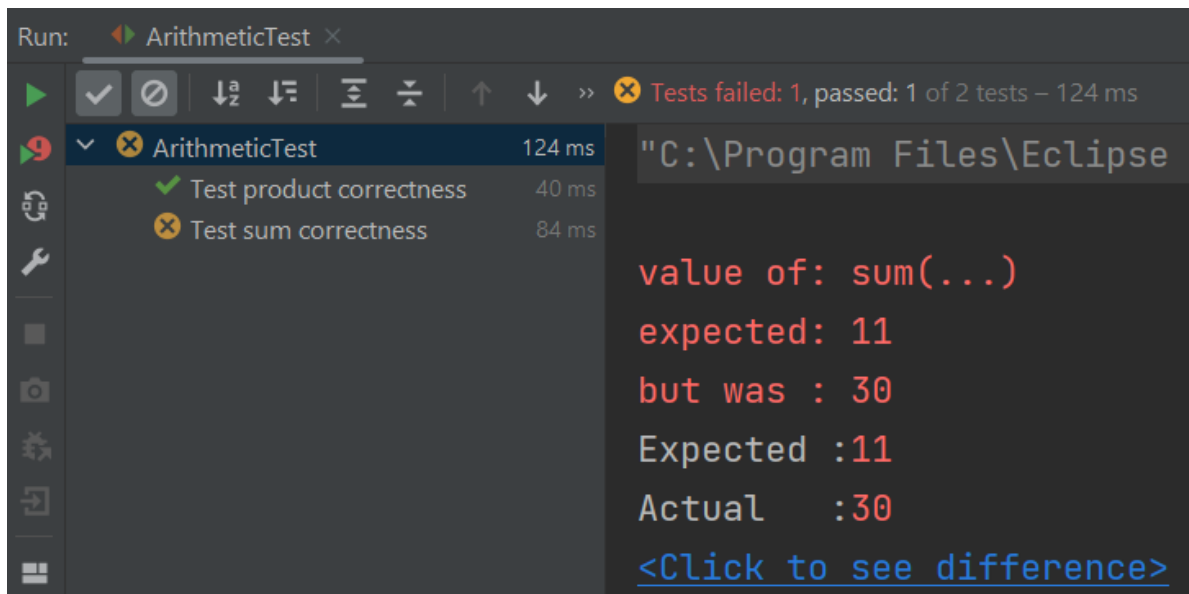


If you follow the prompts, you will (probably) see something wrong! **Don't fix it yet.**

Testing Your Code

While we could run the `Arithmetic` file again and again to check that our code works correctly, it would take a lot of time to type into the program each time, and manually check that the output is correct. Instead, we use **tests**.

Open `ArithmeticTest`, and click the green triangle(s) next to the `public class ArithmeticTest`. This will run the tests that we have provided in this assignment. At this point, you will see the following:



The green checkmark (✓) indicates tests that you have passed, while the yellow X (✗) indicates tests that you have failed. Don't worry about the doubled output; this is a strange quirk of IntelliJ.

i This should look familiar. The test that failed on Gradescope is the same test that we see here! We can run these tests locally, very easily!

✍ Task: Fix the bug in `Arithmetic.java` so that the tests pass.

Saving Your Work using Git and GitHub

As you are making changes to your code, it is good practice to save your work often. We have briefly discussed the commands, but now we will explain how they should be used in practice. In the case that you ever want to go back to another version of your code, it is better to have more options to roll back to. The next set of instructions will talk you through git's version of saving work through snapshots of your file system called commits.

1. After you have made some changes to the code within our local repository, git will take notice of these changes. To see the current state of your local repository, use the command `git status`. Run this and try to interpret the results. Do they make sense to you or do they

match your intuition? It is a good habit to run this before running other git commands to know what the state of things are.

2. To save work that we have completed on a file, we first stage the file, and then we can commit it. We stage a file with the command `git add`. This does not save the file, but it marks it to be saved the next time you commit. The two below commands show what saving work looks like in a git repository. For `git add` depending on what directory you are in, the path to the file you are adding might differ (use `git status` to see the path).

The `-m "Completed Arithmetic.java"` part of the commit command specifies a message to be attached to this snapshot. You should always have a commit message to identify what exactly happened in this commit. As an example workflow:

```
git add lab01/src/Arithmetic.java
git commit -m "lab01: Completed Arithmetic.java"
```

If you run `git status`, you will see that `Your branch is ahead of 'origin/main'`. You will also see that the staged changes are no longer staged, and are instead committed. If you haven't edited since staging, you shouldn't have any changes not staged for commit.


3. We want to push these changes to the GitHub repository so that your changes can be seen by us and Gradescope. Your changes will also be available to `pull`ed if you had your repo initialized in other places or other computers.

```
git push origin main
```

`git status` will now show that `Your branch is up to date with 'origin/main'.`

⚠ Get into the habit of saving your files and doing the `git commit` step *often* (i.e. every 15 minutes). It will be incredibly helpful when you mess things up, since it allows you to back out of changes and to see what you have changed recently.

Basically, right when you sit down to work in your repository, first `git pull` to make sure you are starting with the most recent code. While you are working, frequently commit. When you are finished, `git push` so all your changes are uploaded and ready for you to pull again next time.

 **Task:** Follow the instructions (again!) in the [Submitting to Gradescope section of the Assignment Workflow guide](#) to submit to Gradescope. This time, you should receive a full score on the lab.

Deliverables

As a reminder, this assignment has an [FAQ page](#). There are two required files, all inside the `lab01` directory:

`magic_word.txt`

You should have received the correct magic word from completing the git checkoff.

`Arithmetic.java`

You should have fixed the bug so that the tests pass. We check this file with an autograder! For this lab, the autograder tests are the same as the ones you have on your computer.

Be sure to submit **again** according to the [submission section](#), so that you submit your completed lab. Congratulations on finishing your first CS 61B lab!

Optional: Josh Hug's Color Scheme IntelliJ Download

Per Josh Hug:

I'm not a big fan of the default IntelliJ colors, so I made my own color scheme, which is a very close imitation of the extremely good Sunburst theme for Sublime. To use my theme, download

[hug_sunburst](#), and import it using the “File → Manage IDE Settings → Import Settings” option in IntelliJ. You might end up with large text, which I use for recording videos. To adjust the size of the font in IntelliJ to your liking, see [this link](#).

Last built: 2023-10-26 18:40 UTC