

Juhan Hong GE01 Python, Pair Programming and Version Control

Names of the person you collaborated

Tandon

Description: Learning how to learn new technologies. This is not about getting everything working perfectly the first time but collaborating, communicating, finding resources and problem solving with others. Most of all do not panic if you run into issues. Note the issues and how you resolved them.

Think about what information is helpful to have for the next time you do this.

Find 4 or more resources that could be valuable for a new person getting started with python and version control.

Brief description	Resource
YouTube video for python for beginner(Video covers everything from variables to OOP)	https://www.youtube.com/watch?v=kqtD5dpn9C8
YouTube video for git and github(Covers many commands that are useful; also video is chaptered for easier access)	https://www.youtube.com/watch?v=tRZGeaHPoaw
CS50 harvard python course(free and covers each topic in detail. Covers topics like Unit Test, Regex, OOP and more)	https://cs50.harvard.edu/python/ https://cs50.harvard.edu/python/2022/weeks/0/
Practice git commands. exercise and practice heavy	https://www.w3schools.com/git/

Start exploring git, github, command line, and python in a virtual environment.

[1 Python and IDE](#)

[Install Python](#)

[Install VS Code IDE](#)

[2 Pair Programming Video](#)

[3 Version Control](#)

[Set-up git and github repository](#)

[Add, Commit, Push Practice](#)

[Branching](#)

[Version Control Concepts](#)

[4 Resume and Interview Questions](#)

1 Python and IDE

Set up your python and IDE for your python development.

Install Python

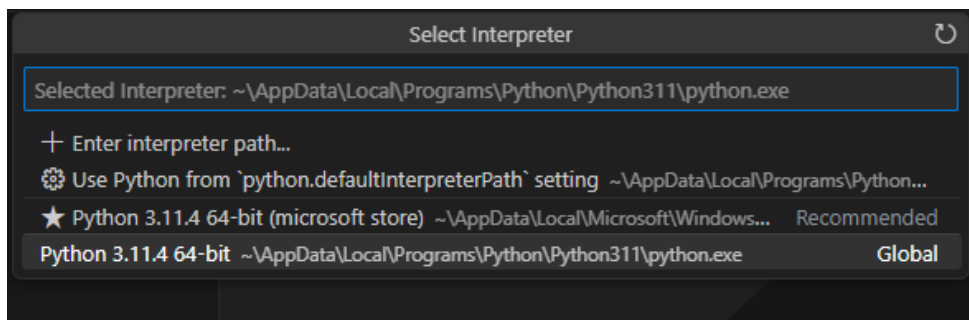
1. Open the command window and check your python version to see if you have it installed.
2. Install python version 3.11 [Download Python](#). If on windows and have older version of python you should uninstall first : [How to Uninstall Python](#)

Install VS Code IDE

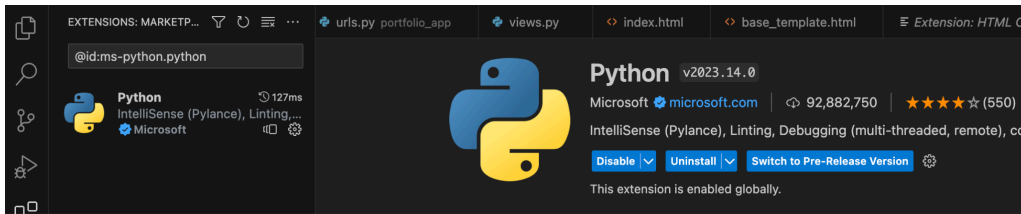
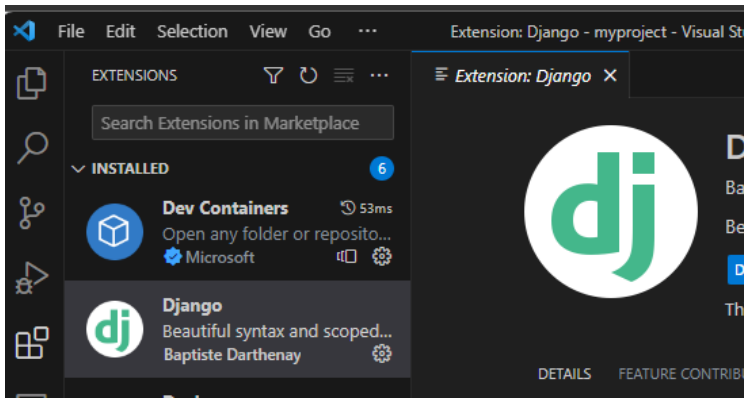
You can use a different IDE but this is what I will be using in my lectures. This has nice tools to integrate with python, django and databases.

<https://code.visualstudio.com/download>

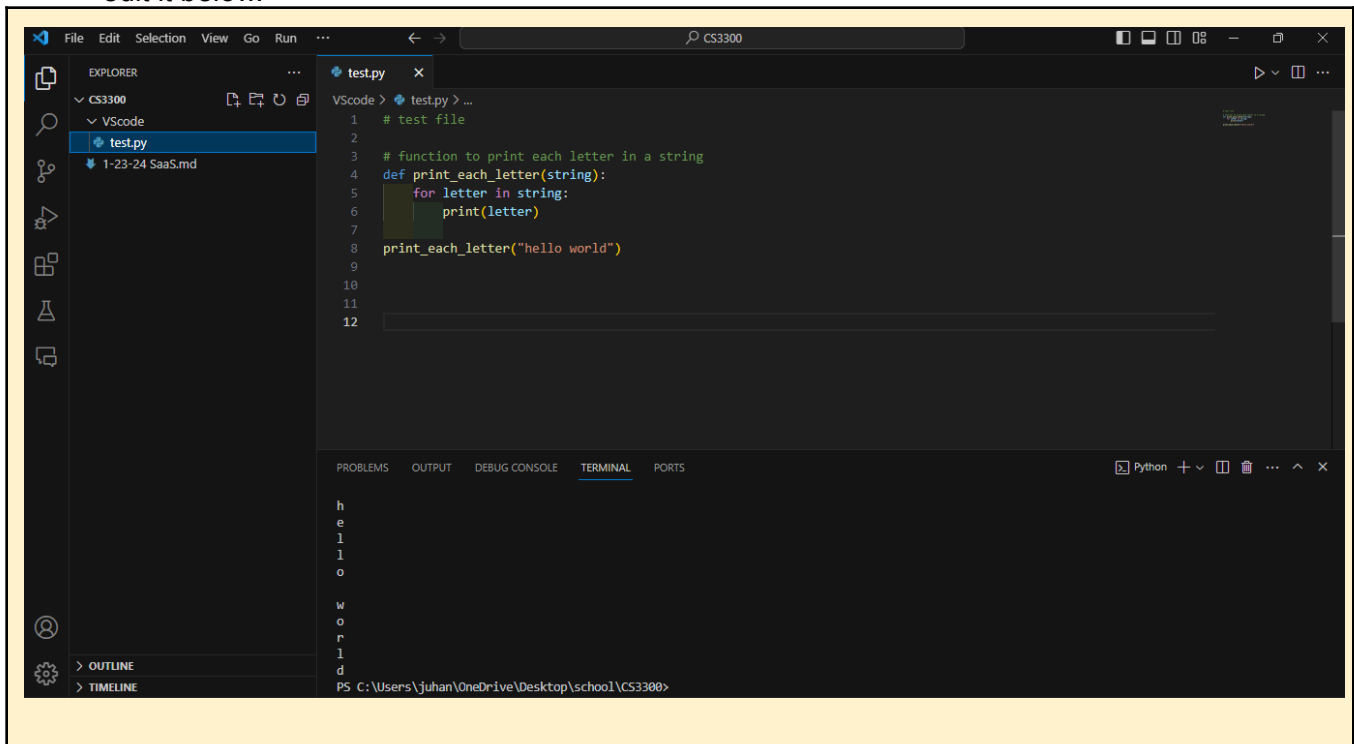
1. Configure the Python interpreter: In Visual Studio Code, open the Command Palette by pressing `Ctrl+Shift+P` (Windows/Linux) or `Cmd+Shift+P` (Mac). Search for "Python: Select Interpreter" and choose the Python interpreter associated with your virtual environment (e.g., `myenv`).

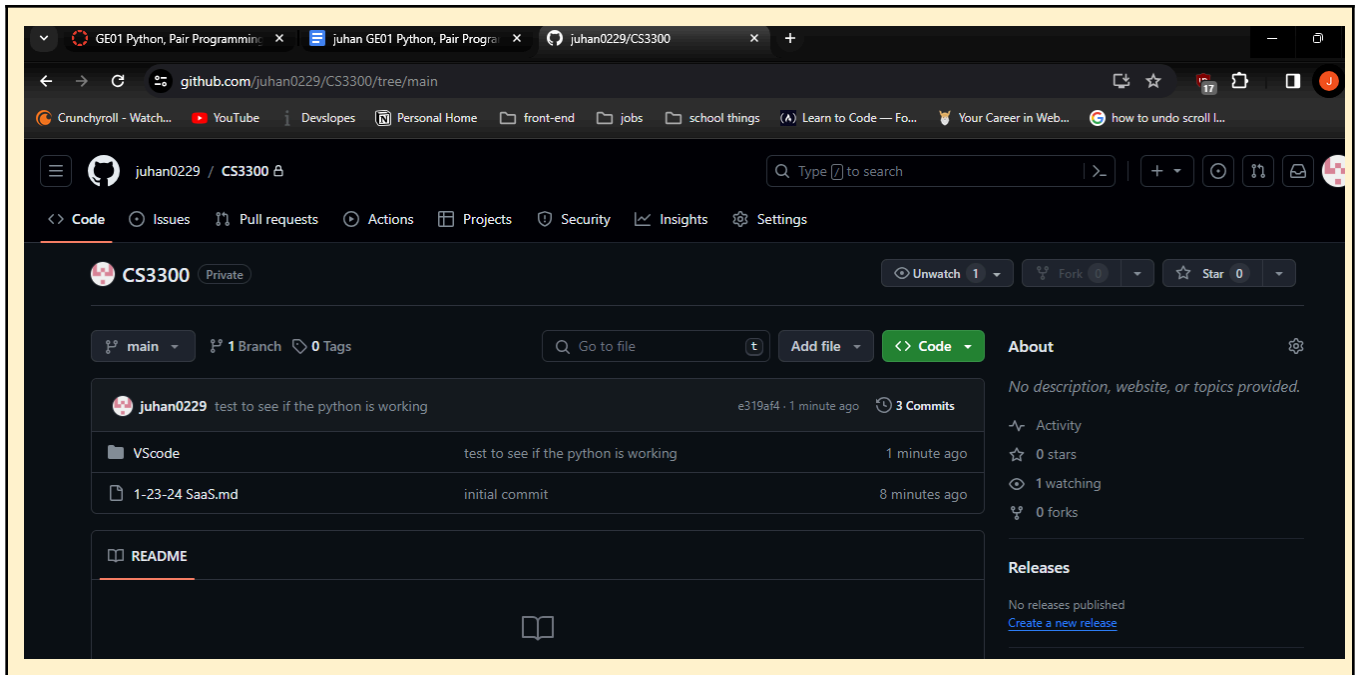


2. Install the Django extension developed by Baptiste Darthenay: In Visual Studio Code, go to the Extensions view and search for the "Django" extension. Install it to benefit from Django-specific features and enhancements for what we will be doing later.



3. You can use this to edit your python file for practice.
4. Take a screenshot of the ide you have set up and the python file from the repository once you edit it below.





2 Pair Programming

Goal: Improve software quality by having multiple people develop the same code.

Setup:

- One shared computer, alternate roles
- Driver: Enters code while vocalizing work
- Observer: Reviews each line as it's typed, acts as safety net + suggest next steps

Effects:

- Cooperative, a lot of talking! + Increases likelihood that task is completed correctly
- Also transfers knowledge between pairs

Start learning the basics by going through [Hello, World! - Free Interactive Python Tutorial](#) by following the instructions below.

- You should spend at least 20 minutes pair programming



- Choose video screen-recording software that you can use to capture your discussion and screen. (such as <https://obsproject.com/>)

Where it says exercise code: that means for that section you are doing the exercise at the end of the information.

- Do not copy the solution code. Instead copy your code and paste below. Add any notes that would be helpful.
- Do not worry if you do not finish all the parts when pair programming but you should start pair programming and videoing with lists.
- Complete on your own after the pair programming ends.

Scan the following sections before pair programming. Take turns summarizing each section to the other. Add any brief notes or examples.

[Hello, World!](#)

To print things in python, it is a much simpler version than any other language ive used. All you got to do is say:

print() and put whatever you want to print in the parenthesis.

[Variables and Types](#)

Tanden explained to me that in python, you don't really need to declare variables with the types, but it is always a good practice to initialize them. For example,

```
myVar = 100
```

That line of code is declared and initialized to be 100 without declaring it being an int.

[Lists](#) Review and complete exercise code:

In List you can use “append” on a list to add things into the list.
From exercise, we did

```
Numbers.append[1]
```

```
Numbers.append[2]
```

To add number 1 and 2 into the list.

You can also assign variables in this fashion:

From exercise

```
Second_string = names[1]
```

The variable second-string will store the second item in the list.

[Basic Operators](#) Review and complete exercise code:

Scan the following sections. Take turns summarizing each section to the other. Add any brief notes or examples.

Basic Operators:

Basic operations in python perform very similar to other languages such as C or java.

String Formatting:

String formatting in python works very similar to C. Meaning that we can use %s as the format specifier to print strings.

Basic String Operations:

There are many useful string operations in python:

- `len()`: gets the length of the string
- `string.index("")`: gets the index position of the wanted char
- `string.count("")`: counts the instances of the char in the string
- `string[#:#]`: slices the string
- `String[::-1]`: reverse the string
- `String.lower()` / `string.upper()`
- `string.startswith()` / `string.endswith()`
- `string.split (" ")`: splits the string into more strings grouped in a list.

Conditions:

Conditional statements in python are very similar to other languages but look more digestible to non programmers by using words instead of symbols. For example:

If `age == 10` and `name == "john"`:

Compared to in C:

```
if(age == 10 && name == "john"){
```

Loops:

Loops in python also works very similarly to other languages but with a few changes.

For for loops, it looks much simplified by just saying: `for x in range(#):`

For while loops, also looks much more simple by just saying: `while number < #:`

Functions Review and complete exercise code:

Functions in python works very similarly to C and java as well.

Use "def" to create a function in python:

```
def some_func(parameters):
```

call functions by some_func()

```
def name_the_benefits_of_functions():  
    list_of_benefits = list_benefits()  
    for benefit in list_of_benefits:  
        print(build_sentence(benefit))
```

[Classes and Objects](#) Review and complete exercise code:

Classes and object in python works very similar to Java. We can call each objects with a dot operator. The variables can also be updated simply with the = operator. In the exercise, it reviewed creating a new object, car1 and car2 and assigning variables.

```
# your code goes here  
car1 = Vehicle()  
car2 = Vehicle()  
|  
car1.name = "Fer"  
car1.color = "red"  
car1.value = 60000.00  
  
car2.name = "Jump"  
car2.color = "blue"  
car2.value = 10000.00
```

[Dictionaries](#) Review and complete exercise code:

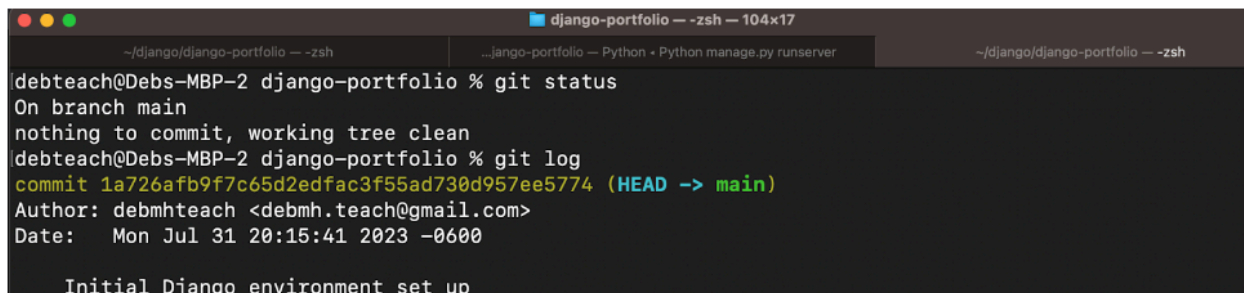
From the website, dictionaries are just like arrays but instead of working with indexes, it works with keys and values. The practice example on the website used a phonebook as example. Phonebook, containing the name(key) and the numbers(values). We can add and remove elements with these commands:

```
# your code goes here  
  
phonebook["Jake"] = 938273443|  
  
phonebook.pop("Jill")
```

3 Version Control

Set-up git and github repository

Use the command line tool of your preference in your environment. I ended up using command prompt on my windows but also have used windows powershell.I use the generic command tool on my mac. Here is an example of using the default command prompt

A terminal window titled 'django-portfolio -- zsh -- 104x17' showing the output of 'git status' and 'git log' commands. The 'git status' output indicates the user is on the 'main' branch and the working tree is clean. The 'git log' output shows a single commit with hash '1a726afb9f7c65d2edfac3f55ad730d957ee5774' on the 'main' branch, authored by 'debmhteach' on 'Mon Jul 31 20:15:41 2023 -0600'. At the bottom, it says 'Initial Django environment set up'.

```
debmhteach@Debs-MBP-2 django-portfolio % git status
On branch main
nothing to commit, working tree clean
debmhteach@Debs-MBP-2 django-portfolio % git log
commit 1a726afb9f7c65d2edfac3f55ad730d957ee5774 (HEAD -> main)
Author: debmhteach <debmh.teach@gmail.com>
Date: Mon Jul 31 20:15:41 2023 -0600

Initial Django environment set up
```

Research

- What is git and github? What does git provide? What does github provide?
- How can you create a github repository from a local folder?
- What documentation could be useful to help understand the commands?

Include resources in the table above.

1. Create a python file in a local folder cs3300-version-practice
2. Create a folder called documentation in cs3300-version-practice that contains this document.
3. Create a github account if you do not have one.
4. Create a github repository that is public from the local folder.

Explain what you did and the commands you used.

1. To create a local folder, I have used the command 'cd' to find the location. Once i was at the right place, in my case it was CS3300 folder, I created a new folder with this command: mkdir "cs3300-version-practice". Used the command: cd cs3300-version-practice to go into the folder. From there I used the command: touch testfile.py to create a python file in the new directory.


```

juhan@LAPTOP-OM18U1KA MINGW64 ~/OneDrive/Desktop/school/cs3300-version-practice
(main)
$ ls
anothertest.py  documentation/  new.py  testfile1.py

juhan@LAPTOP-OM18U1KA MINGW64 ~/OneDrive/Desktop/school/cs3300-version-practice
(main)
$ touch testfile3.py

juhan@LAPTOP-OM18U1KA MINGW64 ~/OneDrive/Desktop/school/cs3300-version-practice
(main)
$ ls
anothertest.py  documentation/  new.py  testfile1.py  testfile3.py

```

2. While still in cs3300-version practice directory, used the command: mkdir documentation to create a directory.

```

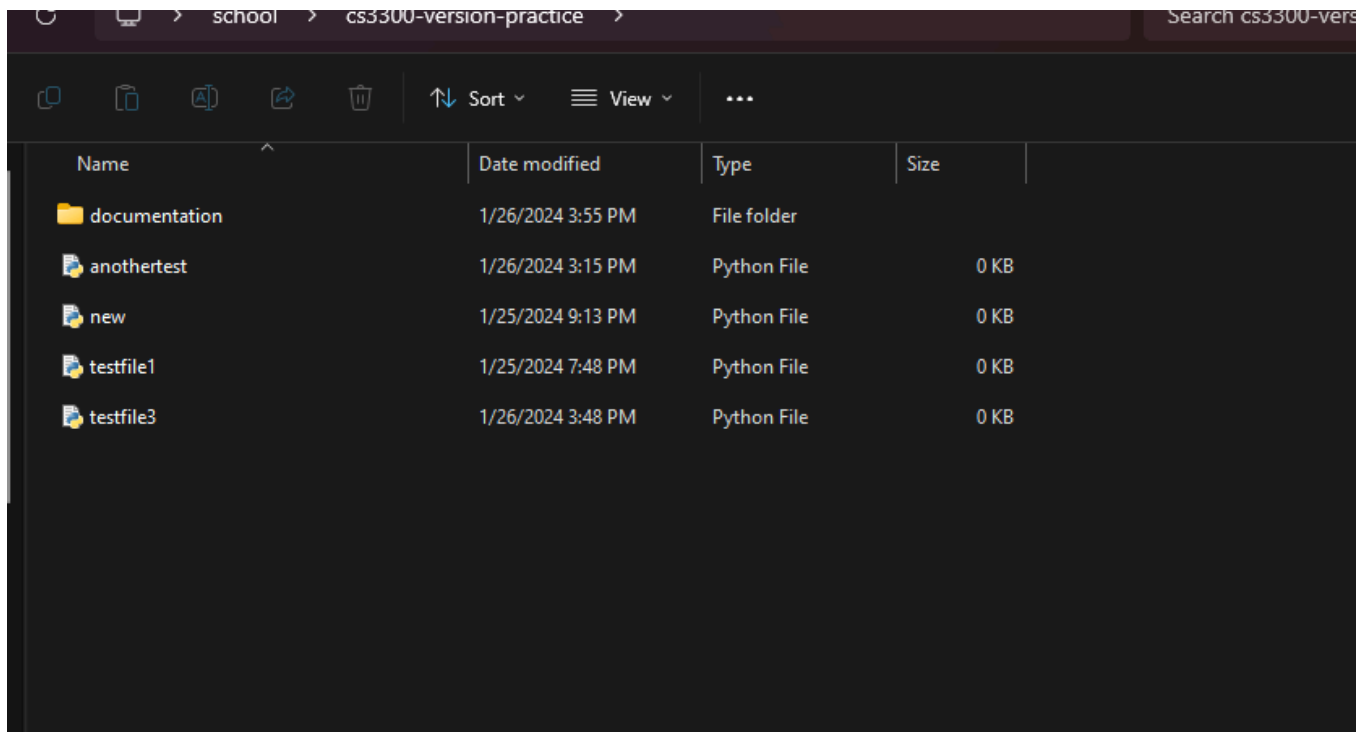
juhan@LAPTOP-OM18U1KA MINGW64 ~/OneDrive/Desktop/school/cs3300-version-practice
(main)
$ mkdir documentation

juhan@LAPTOP-OM18U1KA MINGW64 ~/OneDrive/Desktop/school/cs3300-version-practice
(main)
$ ls
anothertest.py  documentation/  new.py  testfile1.py

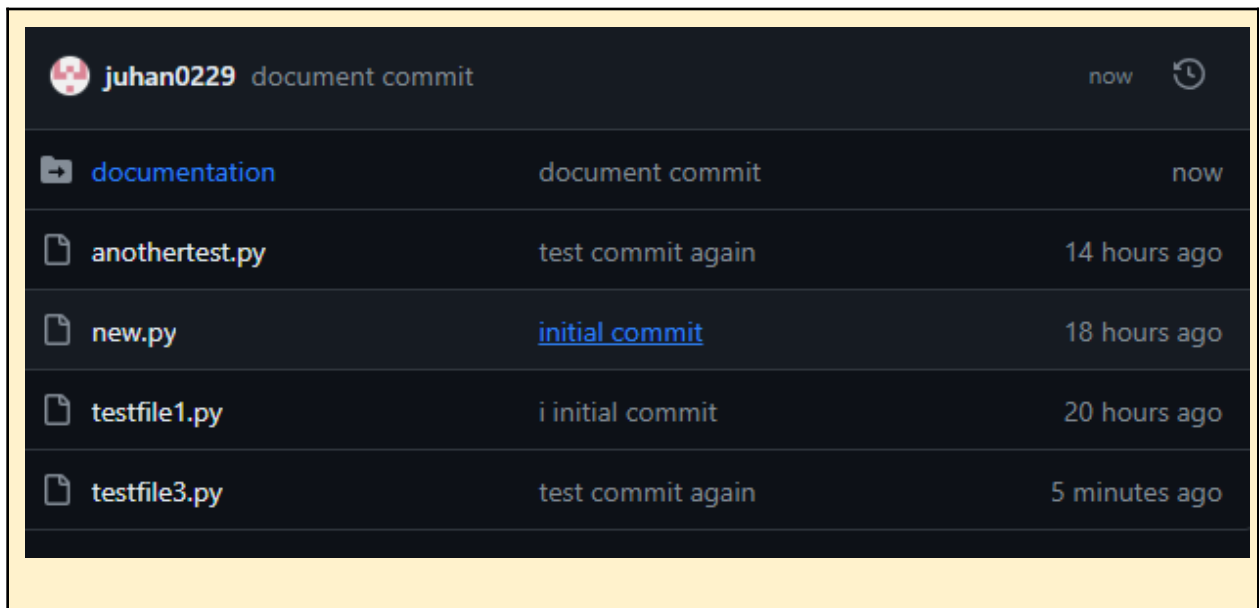
```

3. To create a github account, simply go to github website at github.com and sign up.
 - a. I used my personal email in order to keep using them after school.
4. In github, you can create a public repository. After creating a repository, in git you can set up a remote access with these following commands:
 - a. Git init (initialize the directory)
 - b. Git remote add origin
<https://github.com/juhan0229/cs3300-version-practice>(this is the url for the repository on github. Or whichever repository you want remote access to via git.)
 - c. Git branch -M main
 - d. Git push -u origin main (push the repo onto github)

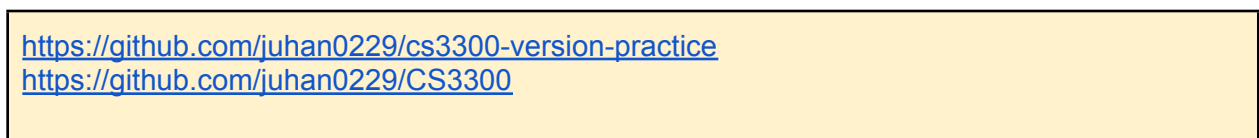
Paste a screenshot of your local directory code



Paste a screenshot of your github repository code



Paste the url to you github repository code



5. You may need to generate an SSH Key pair to configure remote access to your repositories. Github's instructions for this process can be found [here](#).
6. You may need to set

```
git config --global user.email "you@email"
```

 (email associated with repository)

```
git config --global user.name "Your Name"
```

Add, Commit, Push Practice

1. You can just work with updating a python file.
 1. Check the git branch and status

```
git branch  
git status
```

2. Update the file. Before you can commit the version you must add the new file to the index (the staging area)

```
git add .  
git status
```

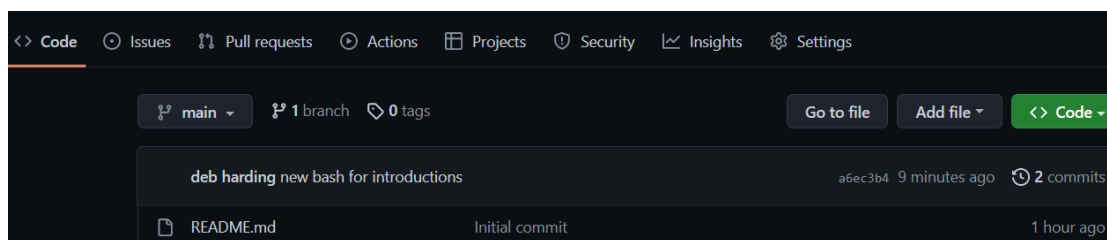
3. Record changes to the local repository with a description but first you might need to include the author identity. Then check the status

```
git commit -m 'add description'  
git status
```

4. You will add your code, commit and push. Then explore the repository on the remote server, github

```
git push
```

```
git status
```



Branching

1. From the command line in your repository on your computer check the log and what branch you are on.
 2. Create a branch called sprint01 and check the log and branch
- Copy and paste the commands you used

```
juhan@LAPTOP-OM18U1KA MINGW64 ~/OneDrive/Desktop/school/CS3300 (main)
$ git branch
* main

juhan@LAPTOP-OM18U1KA MINGW64 ~/OneDrive/Desktop/school/CS3300 (main)
$ git checkout sprint01
error: pathspec 'sprint01' did not match any file(s) known to git

juhan@LAPTOP-OM18U1KA MINGW64 ~/OneDrive/Desktop/school/CS3300 (main)
$ git checkout -b sprint01
Switched to a new branch 'sprint01'

juhan@LAPTOP-OM18U1KA MINGW64 ~/OneDrive/Desktop/school/CS3300 (sprint01)
$ git status
On branch sprint01
nothing to commit, working tree clean

juhan@LAPTOP-OM18U1KA MINGW64 ~/OneDrive/Desktop/school/CS3300 (sprint01)
$ git branch
main
* sprint01

juhan@LAPTOP-OM18U1KA MINGW64 ~/OneDrive/Desktop/school/CS3300 (main)
$ git log
commit f6ec4954c90f84084f1d86cf0179198e619c6961 (HEAD -> main, origin/main, sprint01)
Author: juhan hong <juhan.hong0@gmail.com>
Date: Fri Jan 26 16:35:25 2024 -0700

    added a few lines

commit 4f343fef4d835728322d88a372f49805341aff36
Author: juhan0229 <juhan.hong0@gmail.com>
Date: Fri Jan 26 16:32:12 2024 -0700

    removed a file

commit 0cc4852821ca9210c0e0a7ee172fad98ae87c5bc
Author: juhan0229 <juhan.hong0@gmail.com>
Date: Fri Jan 26 16:25:58 2024 -0700

    commit
```

3. Switch to sprint01 branch to check out code:

```
git checkout 'sprint01'
git branch
git status
```

4. Modify python file and Add the file to the staging area and update the version in your local directory.

Copy and paste the command(s) you used

```
juhan@DESKTOP-53RQ2HD MINGW64 ~/OneDrive/Desktop/school/CS3300 (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean

juhan@DESKTOP-53RQ2HD MINGW64 ~/OneDrive/Desktop/school/CS3300 (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   VScode/pairProgramming.py

no changes added to commit (use "git add" and/or "git commit -a")

juhan@DESKTOP-53RQ2HD MINGW64 ~/OneDrive/Desktop/school/CS3300 (main)
$ git add .

juhan@DESKTOP-53RQ2HD MINGW64 ~/OneDrive/Desktop/school/CS3300 (main)
$ git commit -m "modified python file and update version"
[main 8fbb002] modified python file and update version
1 file changed, 7 insertions(+), 1 deletion(-)

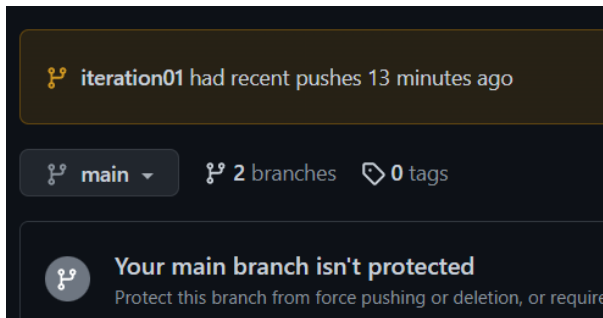
juhan@DESKTOP-53RQ2HD MINGW64 ~/OneDrive/Desktop/school/CS3300 (main)
$ git push -u origin main
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 6 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 492 bytes | 492.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/juhan0229/CS3300
   d48a5b0..8fbb002  main -> main
branch 'main' set up to track 'origin/main'.

juhan@DESKTOP-53RQ2HD MINGW64 ~/OneDrive/Desktop/school/CS3300 (main)
$
```

In the first part, i checked with git status command and it showed that there is no update. Then i added a few lines of code in the python file. I then used git status and it showed that it was modified. The command git add . puts the file in the staging area. Git commit -m "my message" command committed the change, the i pushed it to github with git push -u origin main.

5. Share the changes with the remote repository on the new sprint01 branch. Go to your github and you will see you now have two branches. Click to view the branches. Now others working on the branch could pull your updates from the sprint01 branch.

git push --set-upstream origin sprint01
git status
git log



6. Switch to the main branch and update the remote main branch repository with the change from sprint01 branch. Then go to github to see the versioning.

Copy and paste the commands you used

```
juhan@LAPTOP-OM18U1KA MINGW64 ~/OneDrive/Desktop/school/CS3300 (sprint01)
$ git status
On branch sprint01
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   testfile.py

no changes added to commit (use "git add" and/or "git commit -a")

juhan@LAPTOP-OM18U1KA MINGW64 ~/OneDrive/Desktop/school/CS3300 (sprint01)
$ git add .

juhan@LAPTOP-OM18U1KA MINGW64 ~/OneDrive/Desktop/school/CS3300 (sprint01)
$ git push --set-upstream origin sprint01
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'sprint01' on GitHub by visiting:
remote:   https://github.com/juhan0229/CS3300/pull/new/sprint01
remote:
To https://github.com/juhan0229/CS3300.git
 * [new branch]      sprint01 -> sprint01
branch 'sprint01' set up to track 'origin/sprint01'.

juhan@LAPTOP-OM18U1KA MINGW64 ~/OneDrive/Desktop/school/CS3300 (sprint01)
$ git status
On branch sprint01
Your branch is up to date with 'origin/sprint01'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   testfile.py
```

```

juhan@LAPTOP-OM18U1KA MINGW64 ~/OneDrive/Desktop/school/CS3300 (sprint01)
$ git log
commit f6ec4954c90f84084f1d86cf0179198e619c6961 (HEAD -> sprint01, origin/sprint01, origin/main, main)
Author: juhan hong <juhan.hong0@gmail.com>
Date: Fri Jan 26 16:35:25 2024 -0700

    added a few lines

commit 4f343fef4d835728322d88a372f49805341aff36
Author: juhan0229 <juhan.hong0@gmail.com>
Date: Fri Jan 26 16:32:12 2024 -0700

    removed a file

commit 0cc4852821ca9210c0e0a7ee172fad98ae87c5bc
Author: juhan0229 <juhan.hong0@gmail.com>
Date: Fri Jan 26 16:25:58 2024 -0700

    commit

commit 31487f837bba9631c67b614104cb7fdbb822857
Author: juhan0229 <juhan.hong0@gmail.com>
Date: Fri Jan 26 16:17:47 2024 -0700

    initial commit

juhan@LAPTOP-OM18U1KA MINGW64 ~/OneDrive/Desktop/school/CS3300 (sprint01)
$ git checkout main
Switched to branch 'main'
M    testfile.py
Your branch is up to date with 'origin/main'.

```

I modified the code in the VScode environment while in sprint01 branch.

7. Tag the main branch 'v1.0' then view the tag and push to the remote repository. When you go to the remote repository you should see the tag listed.
Copy and paste the commands you used

```

juhan@LAPTOP-OM18U1KA MINGW64 ~/OneDrive/Desktop/school/CS3300 (main)
$ git merge sprint01
Already up to date.

juhan@LAPTOP-OM18U1KA MINGW64 ~/OneDrive/Desktop/school/CS3300 (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   testfile.py

juhan@LAPTOP-OM18U1KA MINGW64 ~/OneDrive/Desktop/school/CS3300 (main)
$ git tag -a v1.0 -m "version 1 tag"

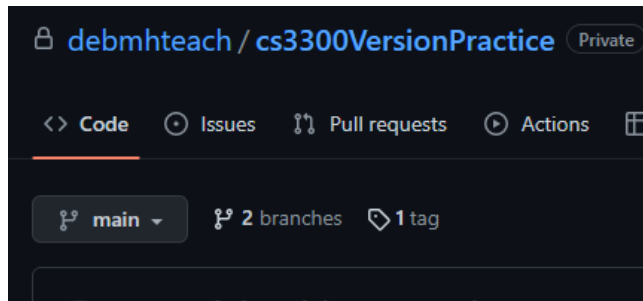
juhan@LAPTOP-OM18U1KA MINGW64 ~/OneDrive/Desktop/school/CS3300 (main)
$ git commit -m "commit with v1.0 tag"
[main e7ab3b2] commit with v1.0 tag
1 file changed, 5 insertions(+), 1 deletion(-)

juhan@LAPTOP-OM18U1KA MINGW64 ~/OneDrive/Desktop/school/CS3300 (main)
$ git push -u origin main

```

For adding a tag, i used `git tag -a v1.0 -m "version 1 tag"`
Then I committed and pushed.

For example



Version Control Concepts

Individually answer each question in your own words, **including any resources you used to help you above**. This will be helpful when you keep technical documentation with your team. **You can use AI to help you understand but answer in your own words.**

3.1 Explain software version control. Address in your description branches, commits, merges, tags.

Software version control works a crucial role when creating and working on software. To put it simply, through version control, collaboration with others are possible since everyone will be looking and working on the same thing. Version control manages changes to a project's source code over time.

Branches: Branches are separate copy of your work that is not related to the repository. You can create as many branches as you want. It will not be merged or committed to the main branch, which is the original source code, unless you want it to. Once you create a branch, you make an exact copy of the project/program that you can add more, delete, etc. If there are bugs or too many mistakes, you can simply delete the branch and it will not affect the main branch.

create a new branch: `git checkout -b "branch name"`

Check what branches you have/branch you're on: `git branch`

Switch branch: `git checkout "branch you want to go to"`

Delete branch: `git branch -D "branch to delete"`

Commits: commits are a snapshot of what you have been working on at the time. Committing often helps you keep track of what kind of changes you've made over time. If there are some serious mistakes, you can see where it went wrong through the history of your commits.

Git commit -m "message about the commit"

Merge: Merge means to combine a certain branch with the main branch. After working on a branch and it is working and ready to be committed and pushed on to the main branch, it can be merged to make the main branch to change inherit the branch you've been working on.

Merge branch: `git merge "branch to merge"`

Tags: Tags means adding the marks to important points in the development history. This keeps track of where the development of the program is.

Add tags: `git tag -a "tag name" -m "message regarding the tag"`

Source: <https://education.github.com/git-cheat-sheet-education.pdf>

3.2 Research what Git is and what its relationship is to software version control. Include how GitHub integrates with git.

Git plays a crucial role in version control. Essentially, it allows the developer to clone, work, and push back onto the main repository. By cloning, all the developers will get the same versions that are live at the time. Git also has so many other functions such as commit, tags, branches, and many others to help developers keep track of everything they are working on for that project.

You can integrate git and github together for some powerful version control combo. I personally have not worked in the industry, but as far as personal use goes, I can work on my desktop, commit and push to github using git, then go on my laptop anywhere, pull, work, commit, push back into the github.

3.2 Explain the following commands and include examples: commit, pull, push, add, clone, status, log, checkout

Git commit -m "message": commits changes to the local environment/repository. Creating history of changes that have been made.

Git pull origin main/"or other branch": pulls or fetches changes from the remote repository and puts it on the main/branch you want to work on.

Git push origin/"or other branch": push or upload the changes to the remote repository from the main/branch you want to push.

Git add "file name": puts the file you want to commit into the staging area.

Git add . : puts all the files in the current local repository with changes into the staging area.

Git clone "repo url": clones or copy the remote repository into the local repository.

Git status: shows the status for all the changes that are not yet in the staging area. Shows up in red if not added to the staging area, and green if they are.

Git log: display the commit history with all the details.(helpful if you commit regularly.)

Git checkout "branch name": switch between branches.

3.3 Explain the difference between a branch and a tag.

Branch simply put is just a working copy of the repository that you are working on. Branches are dynamic, can be created, deleted, switch between, and merge between other branches.

Tags are different than branches since tags are a reference to an important commit. Tags create a fixed point in the project's history.

After working on a branch, you can merge the branch into the main branch, commit with a tag on it and then push it onto the remote repository.

3.4 Describe at least three benefits of a version control system and include an example for each that would be related to industry.

There are a lot of important benefits of version control systems. Here are some i think are important in the industry.

1. Collaboration: As mentioned before, i have not worked in the industry, but when there are many other developers working on the same thing, it is very important that everyone is on the same page and working on the same thing. Proper use of version control will allow all the teammates and everyone else working on the project to look/work on the same thing.

2. Use of branches: Branches allows the developers to create a working copy of the main repository. Meaning, i could have a branch working on a bug fix, meanwhile someone else could have a branch working on a new feature, all while nothing is actually changed on the main repository. When everything is working as intended, then these branches from each respective local repository can be merged on to the main branch then pushed to the remote repository.
3. Commit histories: each developer is going to commit a lot of the changes on their own before pushing it back to the remote repository. Git keeps a very good track of all the commit histories by each developer. When things go wrong, and they will go wrong, it will be easy to look back and track the history to see where exactly things went wrong.

4 Resume and Interview Questions

Create a document that contains the following parts

Part 1: Create a resume to use to interview to be a full stack developer intern that only includes these sections

1. Summary
2. Skills
3. Relevant Experience

Part 2: Interview questions you would ask to see if someone would be a good fit on your team. Include at least 4 questions.