GE01 Python, Pair Programming and Version Control

Effort: Collaborative Assignment CS3300 Academic Integrity (Pairs)

REQUIREMENT: At least 20 minutes of pair programming with someone else.

Points: 40 (see rubric in canvas)

Deliverables: DO NOT UPLOAD A ZIP FILE and submit word or pdf files.

- **Upload this document with your answers**
- **A screencast video of your pair programming activity**
- **Resume and interview questions**

Due Date: See Canvas

Goals:

- Communicate effectively in a variety of professional contexts within a team, with customers, creating oral or written presentations, and technical documents.
- Devotion to lifelong learning: Prepare to learn on their own whatever is required to stay current in their chosen profession, for example, learning new programming languages, algorithms, developmental methodologies, etc.
- Utilize pair programming to begin learning python.

Names of the person you collaborated

Brett Ford, Brennan Romero

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**
Python Tutorial by Developers	The Python Tutorial — Python 3.11.7 documentation
Python Documentation	3.11.7 Documentation (python.org)
Github Tutorial	https://ourcodingclub.github.io/tutorials/git/
Python in VS Code IDE	https://code.visualstudio.com/docs/python/python-tutorial
Command Line Cheatsheet	https://www.git-tower.com/blog/command-line-cheat-sheet

Git Command Line Guide	https://simon-m-mudd.github.io/NMDM_book/#_version_control_with_git
Pro Git Book	https://git-scm.com/book/en/v2

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

1 Python and IDE

Set up your python and IDE for your python development.

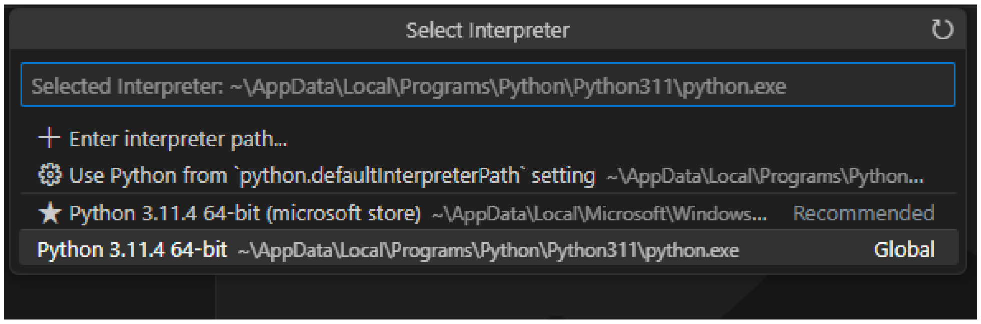
Install Python

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`).



Select Interpreter

Selected Interpreter: ~\AppData\Local\Programs\Python\Python311\python.exe

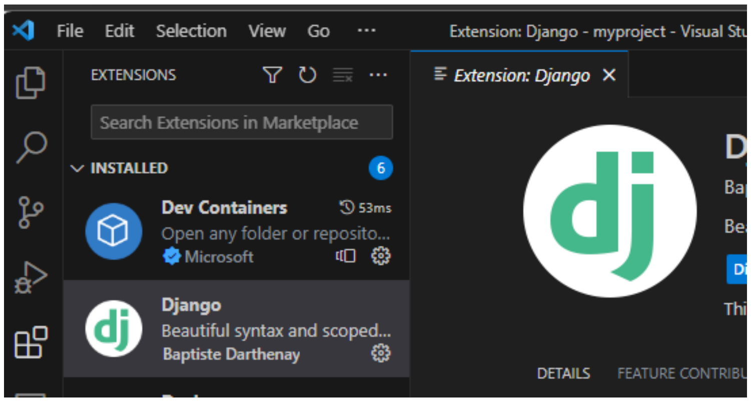
+ Enter interpreter path...

Use Python from `python.defaultInterpreterPath` setting ~\AppData\Local\Programs\Python...

★ Python 3.11.4 64-bit (microsoft store) ~\AppData\Local\Microsoft\Windows... Recommended

Python 3.11.4 64-bit ~\AppData\Local\Programs\Python\Python311\python.exe Global

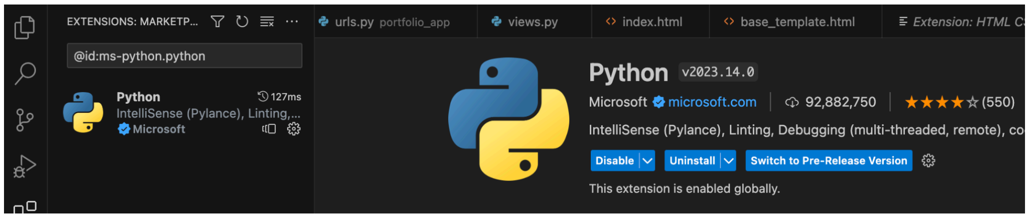
2. Install the Django extension developed by Baptiste Darthenay: In Visual Studio Code, go to the Extensions view and search for the "Django" extension. Install it to benefit from Django-specific features and enhancements for what we will be doing later.



File Edit Selection View Go ···
Extension: Django - myproject - Visual Stu
EXTENSIONS
Extension: Django
Search Extensions in Marketplace
INSTALLED 6
DETAILS
FEATURE CONTRIBU

Dev Containers 53ms
Open any folder or reposito...
Microsoft

Django
Beautiful syntax and scoped...
Baptiste Darthenay

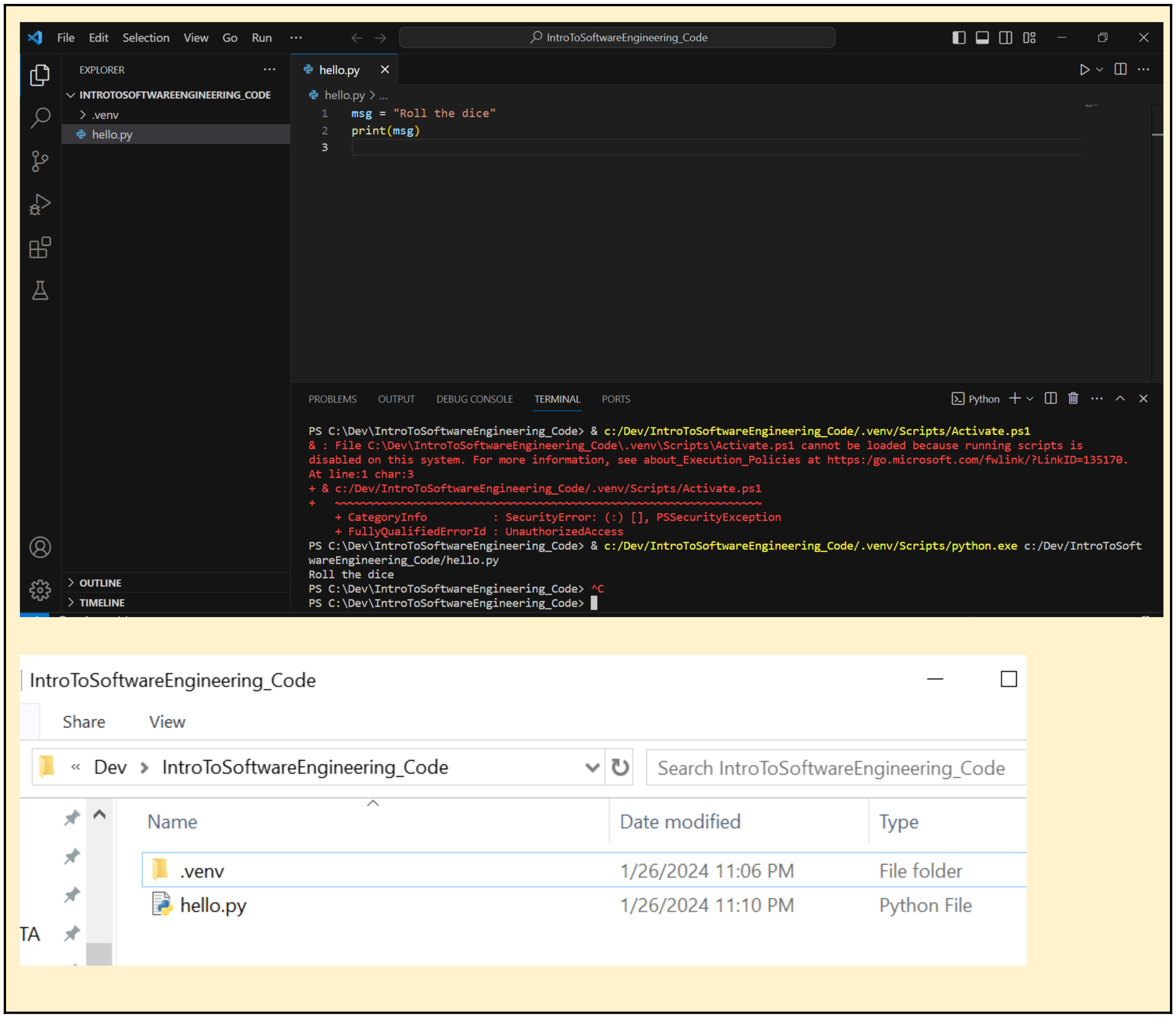


@id:ms-python.python

Python 127ms
IntelliSense (Pylance), Linting,...
Microsoft

Python v2023.14.0
Microsoft microsoft.com | 92,882,750 | ★★★★☆ (550)
IntelliSense (Pylance), Linting, Debugging (multi-threaded, remote), co
This extension is enabled globally.

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.



hello.py > ...

```python
msg = "Roll the dice"
print(msg)
```

```
PS C:\Dev\IntroToSoftwareEngineering_Code> & c:/Dev/IntroToSoftwareEngineering_Code/.venv/Scripts/Activate.ps1
& : File C:\Dev\IntroToSoftwareEngineering_Code\.venv\Scripts\Activate.ps1 cannot be loaded because running scripts is
disabled on this system. For more information, see about_Execution_Policies at https:/go.microsoft.com/fwlink/?LinkID=135170.
At line:1 char:3
+ & c:/Dev/IntroToSoftwareEngineering_Code/.venv/Scripts/Activate.ps1
+   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : SecurityError: (:) [], PSSecurityException
    + FullyQualifiedErrorId : UnauthorizedAccess
PS C:\Dev\IntroToSoftwareEngineering_Code> & c:/Dev/IntroToSoftwareEngineering_Code/.venv/Scripts/python.exe c:/Dev/IntroToSoft
wareEngineering_Code/hello.py
Roll the dice
PS C:\Dev\IntroToSoftwareEngineering_Code> ^C
PS C:\Dev\IntroToSoftwareEngineering_Code> 
```

IntroToSoftwareEngineering_Code
Share
View
« Dev › IntroToSoftwareEngineering_Code
Search IntroToSoftwareEngineering_Code
TA

Name	Date modified	Type
.venv	1/26/2024 11:06 PM	File folder
hello.py	1/26/2024 11:10 PM	Python File

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!** Ex: print("Hello, World!") **Variables and Types** Variables are all objects and do not need to be declared before use or need a declared type

Lists Review and complete exercise code:
Append only takes one argument. 0 based index.
Exercise code:
numbers = []
strings = []
names = ["John", "Eric", "Jessica"]
 	
# write your code here 	
second_name = names[1]
numbers.append(1)
numbers.append(2)
numbers.append(3)
strings.append("hello")
strings.append("world")

Basic Operators Review and complete exercise code:
** is the power operator
Addition operator works with lists.
Lists can be multiplied by an integer to make a repeating sequence
Get length of list with .count()

Exercise code:

```
# TODO: change this code
x_list = [x] * 10
y_list = [y] * 10
big_list = x_list + y_list
```

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

Basic Operators: Section covers arithmetic, string, and list operators.

String Formatting: Section lists basic format specifiers. "%" operator is used to format a set of variables enclosed in a "tuple" (a fixed size list); used along with format specifiers (ex: print("%s is %d years old." % (name, age))).

Basic String Operations: some_string.index(‘some_char’) returns the index of the first occurrence of some_char in some_string. astring.count(‘some-char’) returns how many occurrences of that char there were in the string. Section also shows how to slice strings by index (ex: astring[start:stop:step]), how to reverse strings, how to convert cases, how to test starts and endings of strings, and how to split strings into a list using tokens.

Conditions: Boolean logic is used. "and" and "or" are used not & or ||. “in” operator is covered. Indentation is usually 4 spaces, although tabs and any other space size will work. "is" operator does not match the values of the variables, but the instances themselves. “not” inverses a boolean expression.

If statements look like this:

```
if condition1 and condition2:

    code_block
```

Loops: 2 types: for and while loops. Can use something like this:

```
for x in range(start, end, step):

    Do something
```

For a while loop:

```
while x < 5:
    Do something
```

break is used to exit a for or while loop. continue skips the current block, and returns to the for or while statement. There are else for and else while loops that execute an else clause when the condition is false, unless break is used within the loop.

Functions Review and complete exercise code:
Syntax is as follows:

```
def funct_name:
    first line of block
    4 spaces for the tabs
```

Exercise:

```
# Modify this function to return a list of strings as defined above
def list_benefits():
    return ["More organized code", "More readable code", "Easier code reuse", "Allowing programmers to share and connect code together"]

# Modify this function to concatenate to each benefit - " is a benefit of functions!"
def build_sentence(benefit):
    return benefit + " is a benefit of functions!"

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

name_the_benefits_of_functions()
```

Classes and Objects Review and complete exercise code:
Ex:

```
class MyClass:
    variable = "blah"

    def function(self):
        print("This is a message inside the class.")

myobjectx = MyClass()
```

__init__() is a special function that is called when a class is being initiated, similar to a constructor.

Exercise:
My code for the solution:

```
car1 = Vehicle()
car2 = Vehicle()
car1.name = “Fer”
car1.kind = “convertible”
car1.color = “red”
car1.value = 60000
car2.name = “Jump”
```

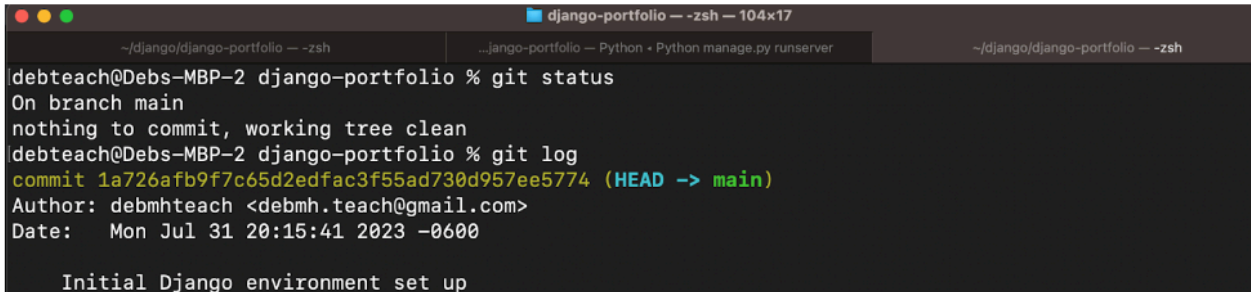
car2.kind = “van”
car2.color = “blue”
car2.value = 10000

Dictionaries Review and complete exercise code:
Similar to arrays but indexed with keys (can be any object). Key corresponds to one value.
Ex:
phonebook = {
    "John" : 938477566,
    "Jack" : 938377264,
    "Jill" : 947662781
}
print(phonebook)
 	
To iterate over key value pairs: for my_key, my_value in phonebook.items():
To remove a pair: del my_dictionary[my_key] or my_dictionary.pop(my_key)
 	
Exercise:
phonebook = {
    "John" : 938477566,
    "Jack" : 938377264,
    "Jill" : 947662781
}
# your code goes here 	
phonebook["Jake"] = 938273443
phonebook.pop("Jill")
 	
# testing code 	
if "Jake" in phonebook:
    print("Jake is listed in the phonebook.")
 	
if "Jill" not in phonebook:
    print("Jill is not listed in the phonebook.")

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


django-portfolio — -zsh — 104×17
~/django/django-portfolio — -zsh
...jango-portfolio — Python ◂ Python manage.py runserver
~/django/django-portfolio — -zsh

```
debteach@Debs-MBP-2 django-portfolio % git status
On branch main
nothing to commit, working tree clean
debteach@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?

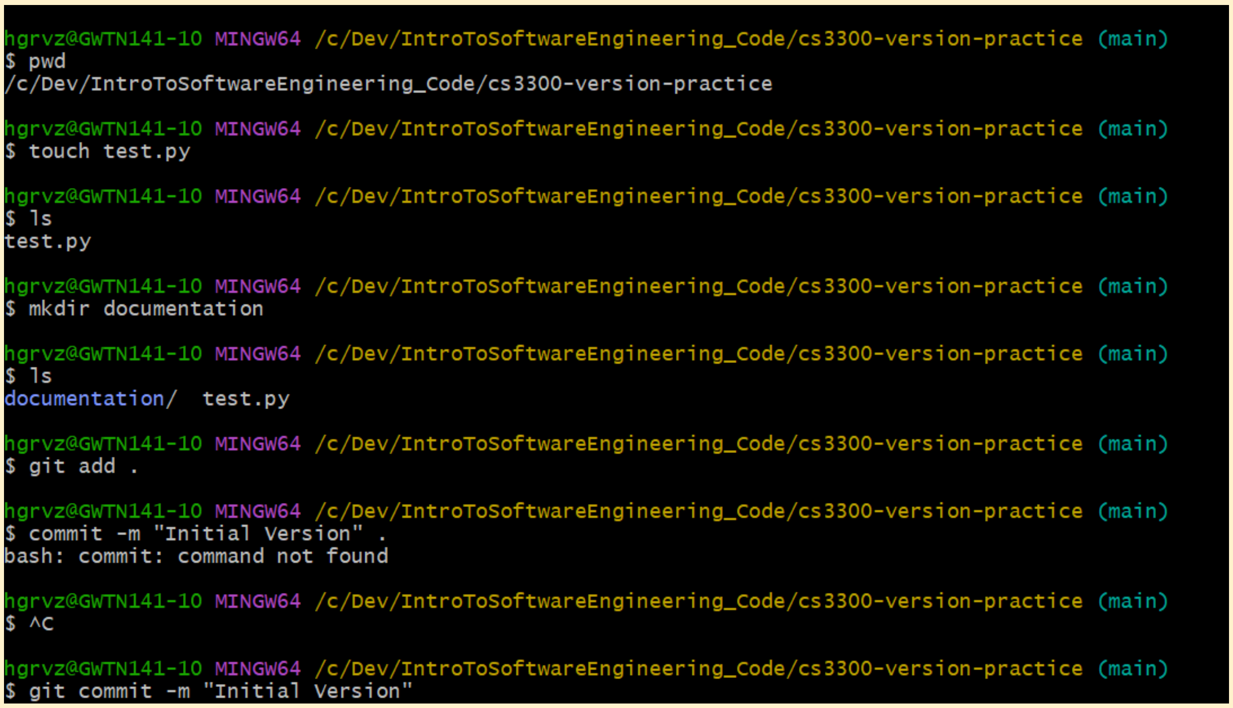
Include resources in the table above.

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.

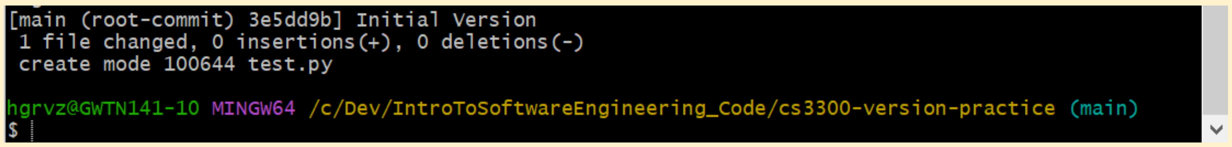
Explain what you did and the commands you used.

I installed Git 2.43.0 after reading the Git Command Line Guide source I listed above. Next, I used file explorer to navigate the my version practice folder. I right-clicked it and selected 'Open Git Bash Here', which opened up Windows Powershell in that directory. Then I used pwd to ensure I was in the local cs3300-version-practice folder. Then I used the command "git init" to initialize my local repository. Next, I used "touch test.py" to create my python file and "mkdir documentation" to create the documentation folder.

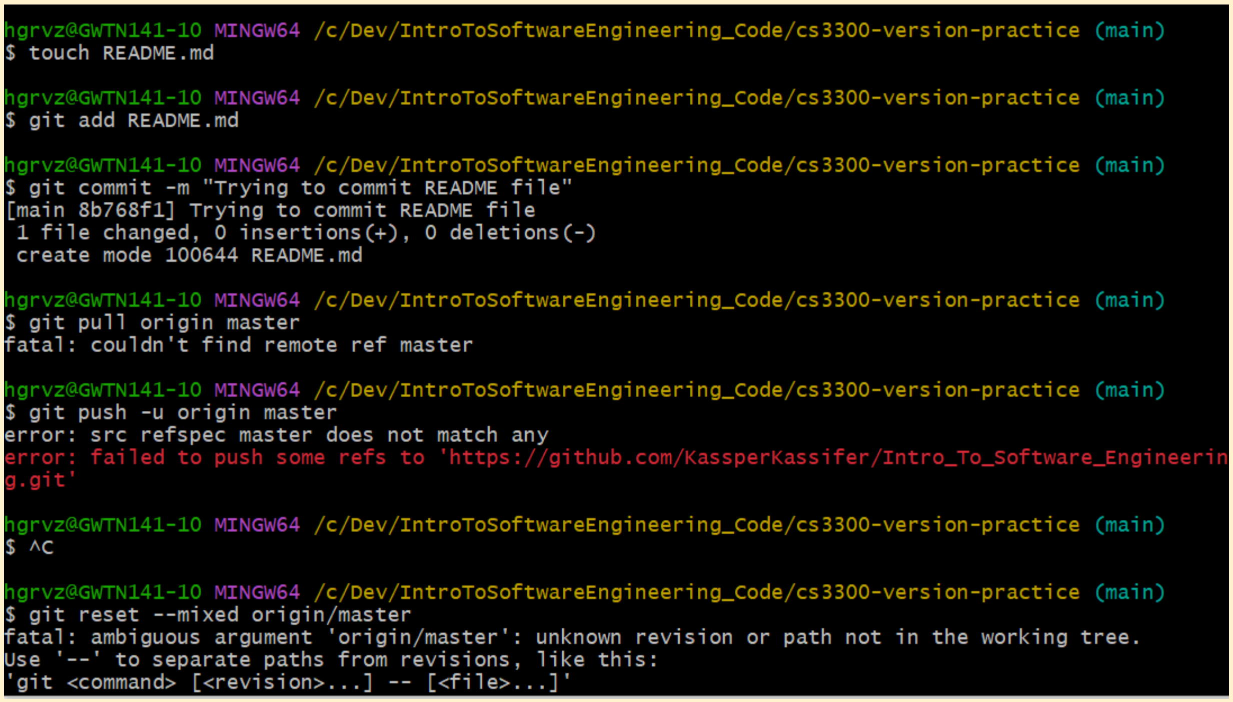
Paste a screenshot of your local directory code



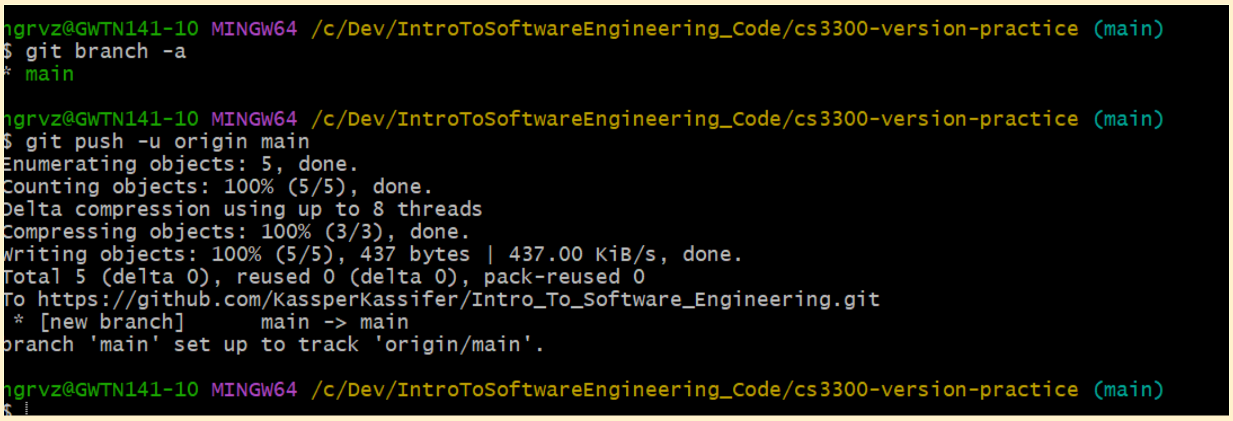
```
hgrvz@GWTN141-10 MINGW64 /c/Dev/IntroToSoftwareEngineering_Code/cs3300-version-practice (main)
$ pwd
/c/Dev/IntroToSoftwareEngineering_Code/cs3300-version-practice

hgrvz@GWTN141-10 MINGW64 /c/Dev/IntroToSoftwareEngineering_Code/cs3300-version-practice (main)
$ touch test.py

hgrvz@GWTN141-10 MINGW64 /c/Dev/IntroToSoftwareEngineering_Code/cs3300-version-practice (main)
$ ls
test.py

hgrvz@GWTN141-10 MINGW64 /c/Dev/IntroToSoftwareEngineering_Code/cs3300-version-practice (main)
$ mkdir documentation

hgrvz@GWTN141-10 MINGW64 /c/Dev/IntroToSoftwareEngineering_Code/cs3300-version-practice (main)
$ ls
documentation/  test.py

hgrvz@GWTN141-10 MINGW64 /c/Dev/IntroToSoftwareEngineering_Code/cs3300-version-practice (main)
$ git add .

hgrvz@GWTN141-10 MINGW64 /c/Dev/IntroToSoftwareEngineering_Code/cs3300-version-practice (main)
$ commit -m "Initial Version" .
bash: commit: command not found

hgrvz@GWTN141-10 MINGW64 /c/Dev/IntroToSoftwareEngineering_Code/cs3300-version-practice (main)
$ ^C

hgrvz@GWTN141-10 MINGW64 /c/Dev/IntroToSoftwareEngineering_Code/cs3300-version-practice (main)
$ git commit -m "Initial Version"
```



```
[main (root-commit) 3e5dd9b] Initial Version
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 test.py

hgrvz@GWTN141-10 MINGW64 /c/Dev/IntroToSoftwareEngineering_Code/cs3300-version-practice (main)
$
```

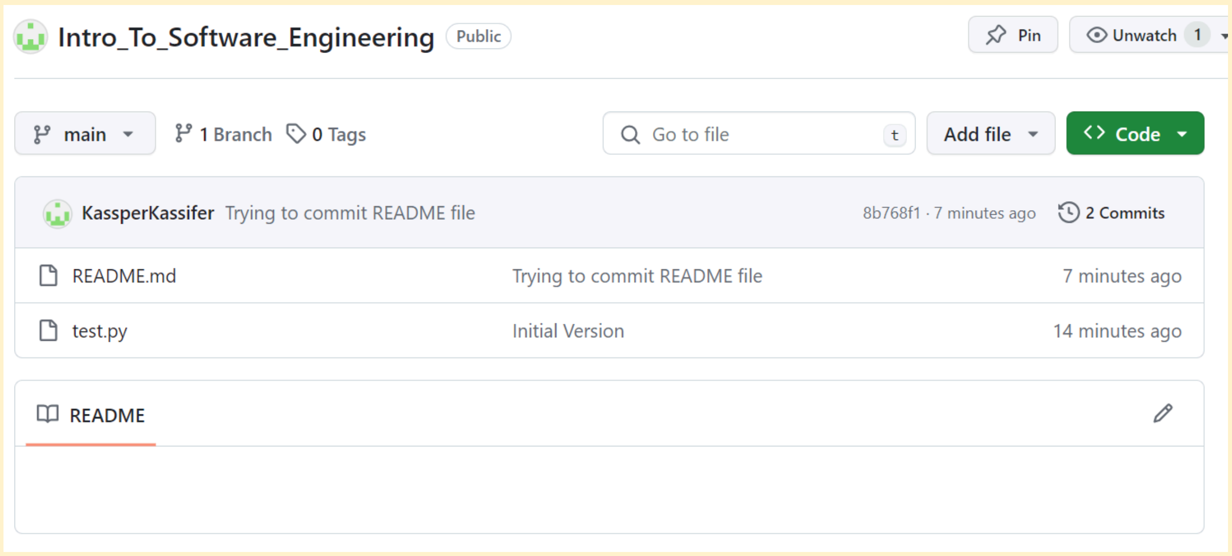


```
hgrvz@GWTN141-10 MINGW64 /c/Dev/IntroToSoftwareEngineering_Code/cs3300-version-practice (main)
$ touch README.md

hgrvz@GWTN141-10 MINGW64 /c/Dev/IntroToSoftwareEngineering_Code/cs3300-version-practice (main)
$ git add README.md

hgrvz@GWTN141-10 MINGW64 /c/Dev/IntroToSoftwareEngineering_Code/cs3300-version-practice (main)
$ git commit -m "Trying to commit README file"
[main 8b768f1] Trying to commit README file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 README.md

hgrvz@GWTN141-10 MINGW64 /c/Dev/IntroToSoftwareEngineering_Code/cs3300-version-practice (main)
$ git pull origin master
fatal: couldn't find remote ref master

hgrvz@GWTN141-10 MINGW64 /c/Dev/IntroToSoftwareEngineering_Code/cs3300-version-practice (main)
$ git push -u origin master
error: src refspec master does not match any
error: failed to push some refs to 'https://github.com/KassperKassifer/Intro_To_Software_Engineerin
g.git'

hgrvz@GWTN141-10 MINGW64 /c/Dev/IntroToSoftwareEngineering_Code/cs3300-version-practice (main)
$ ^C

hgrvz@GWTN141-10 MINGW64 /c/Dev/IntroToSoftwareEngineering_Code/cs3300-version-practice (main)
$ git reset --mixed origin/master
fatal: ambiguous argument 'origin/master': unknown revision or path not in the working tree.
Use '--' to separate paths from revisions, like this:
'git <command> [<revision>...] -- [<file>...]'
```



```
ngrvz@GWTN141-10 MINGW64 /c/Dev/IntroToSoftwareEngineering_Code/cs3300-version-practice (main)
$ git branch -a
* main

ngrvz@GWTN141-10 MINGW64 /c/Dev/IntroToSoftwareEngineering_Code/cs3300-version-practice (main)
$ git push -u origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (5/5), 437 bytes | 437.00 KiB/s, done.
Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/KassperKassifer/Intro_To_Software_Engineering.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.

ngrvz@GWTN141-10 MINGW64 /c/Dev/IntroToSoftwareEngineering_Code/cs3300-version-practice (main)
$
```

Paste a screenshot of your github repository code



KassperKassifer Trying to commit README file 8b768f1 · 7 minutes ago

README.md	Trying to commit README file	7 minutes ago
test.py	Initial Version	14 minutes ago

Paste the url to you github repository code

https://github.com/KassperKassifer/Intro_To_Software_Engineering

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)

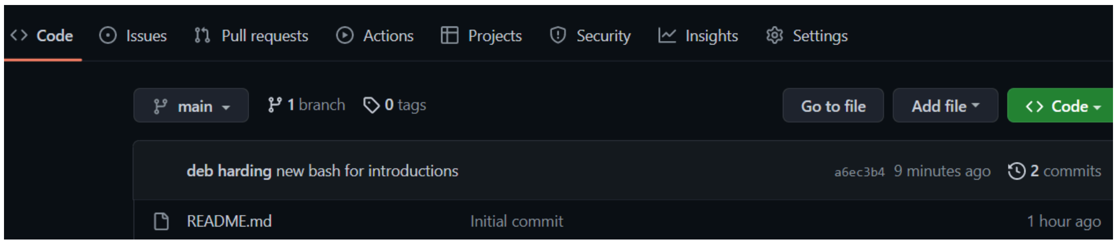
```
git add .
git status
```

    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
```



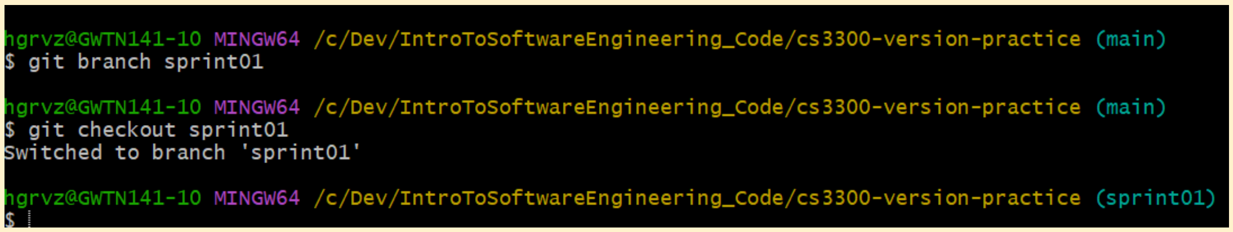
deb harding new bash for introductions a6ec3b4 9 minutes ago 2 commits

README.md Initial commit 1 hour ago

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

I used “git branch” to view my current branch, which was main.
Then I used “git branch sprint01” and “git checkout sprint01”


```
hgrvz@GWTN141-10 MINGW64 /c/Dev/IntroToSoftwareEngineering_Code/cs3300-version-practice (main)
$ git branch sprint01

hgrvz@GWTN141-10 MINGW64 /c/Dev/IntroToSoftwareEngineering_Code/cs3300-version-practice (main)
$ git checkout sprint01
Switched to branch 'sprint01'

hgrvz@GWTN141-10 MINGW64 /c/Dev/IntroToSoftwareEngineering_Code/cs3300-version-practice (sprint01)
$ 
```

3. Switch to sprint01 branch to check out code:

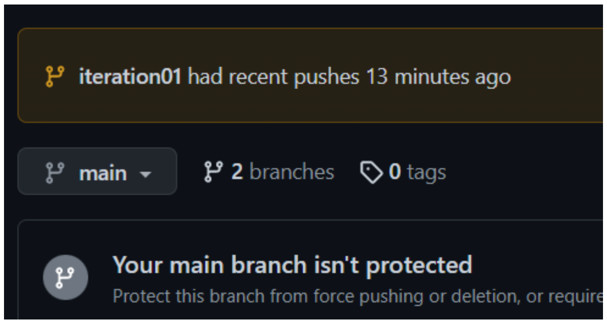
   git checkout 'sprint01'
   git branch
   git status

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

```
$ ls
README.md  documentation/  test.py

hgrvz@GWTN141-10 MINGW64
/c/Dev/IntroToSoftwareEngineering_Code/cs3300-version-practice (sprint01)
$ mv test.py renamed.py

hgrvz@GWTN141-10 MINGW64
/c/Dev/IntroToSoftwareEngineering_Code/cs3300-version-practice (sprint01)
$ ls
README.md  documentation/  renamed.py

hgrvz@GWTN141-10 MINGW64
/c/Dev/IntroToSoftwareEngineering_Code/cs3300-version-practice (sprint01)
$ git add renamed.py

hgrvz@GWTN141-10 MINGW64
/c/Dev/IntroToSoftwareEngineering_Code/cs3300-version-practice (sprint01)
$ git status
On branch sprint01
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   renamed.py

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

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 sprinto1 branch.

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



iteration01 had recent pushes 13 minutes ago

main 2 branches 0 tags

Your main branch isn't protected
Protect this branch from force pushing or deletion, or require

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.

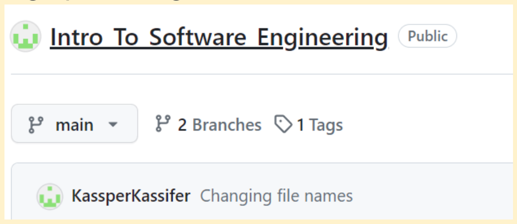
   Copy and paste the commands you used

```
hgrvz@GWTN141-10 MINGW64
/c/Dev/IntroToSoftwareEngineering_Code/cs3300-version-practice (sprint01)
$ git checkout main
Switched to branch 'main'
A       renamed.py
D       test.py
Your branch is up to date with 'origin/main'.

hgrvz@GWTN141-10 MINGW64
/c/Dev/IntroToSoftwareEngineering_Code/cs3300-version-practice (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)
        new file:   renamed.py

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

```
hgrvz@GWTN141-10 MINGW64
/c/Dev/IntroToSoftwareEngineering_Code/cs3300-version-practice (main)
$ git add test.py

hgrvz@GWTN141-10 MINGW64
/c/Dev/IntroToSoftwareEngineering_Code/cs3300-version-practice (main)
$ git commit -m "Changing file names"
[main d01738d] Changing file names
 1 file changed, 0 insertions(+), 0 deletions(-)
 rename test.py => renamed.py (100%)

hgrvz@GWTN141-10 MINGW64
/c/Dev/IntroToSoftwareEngineering_Code/cs3300-version-practice (main)
$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean

hgrvz@GWTN141-10 MINGW64
/c/Dev/IntroToSoftwareEngineering_Code/cs3300-version-practice (main)
$ git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 250 bytes | 250.00 KiB/s, done.
Total 2 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/KassperKassifer/Intro_To_Software_Engineering.git
   8b768f1..d01738d  main -> main
branch 'main' set up to track 'origin/main'.
```

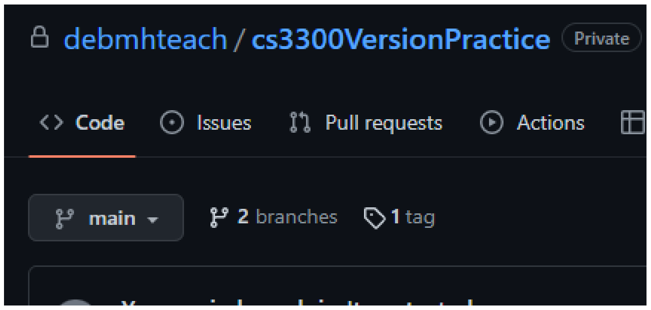
7. Tag the main branch 'v1.0' then view the tag and push to the remote repository. When you go to the remote repository you should see the tag listed.
Copy and paste the commands you used

```
$ git tag -a v1.0
$ git show v1.0
$ git push origin v1.0
```



KassperKassifer Changing file names

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.

Version control software is used to keep track of any changes made to your project. Specifically, it allows you to create versions of code or files that you can access or revert to at any time. It is very helpful for collaboration, especially with the use of branches. Branches are used to do work without messing with the main branch directly. Different people can being doing work on the same project by using multiple branches or sharing branches and pulling in each other's work, all without changing the main, functioning code. Once the side work is functioning and ready to be deployed, you can merge branches back into the main branch. Commits are the tool used to save changes made as a new version in the repository, allowing you to view the committed versions at a later time. You can also use tags such as v1.0 or v.1.5 in your repository to indicate important versions of your program.

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

Git is a distributed version control system that makes it easier for software developers to manage and track their project history. It was made popular due to the speed and expansive capabilities it has with its tools. Git allows users to work on a local repository that they can sync to a remote repository, as well as allows users to view changes before committing code (staging area). GitHub is essentially an extension of Git, as it provides a web-based user interface and access to remote repositories in the cloud. Git is more helpful for managing local files, while GitHub is more helpful for managing finallys remotely.

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

commit: Used to record and permanently save changes in the local repository. Ex: $ git commit -m "Initial version" -> this would commit a version in the local repository with the message given.
pull: Used to get changes from a remote repository and merge them into your current branch. Ex: $ git pull origin sprint01 -> Pulls changes from the sprint01 branch from the origin remote repository.
push: Used to upload changes made in a local repository to a remote repository. Ex: $ git push origin sprint01 -> uploads the changes made locally in the sprint01 branch to the origin remote repository.
add: Used to stage changes made before committing them. Allows you to select what changes to commit. Ex: $ git add test.py -> would add the changes made to the test.py file to the staging area
clone: Used to create a copy of a remote repository to a local machine. Ex: git clone <(opt) -b branch_name> <repository URL> <local_directly> -> would copy a specific branch or the entire repository from the choses URL into the specified local file directory.
status: Used to show if changes in the current branch are untracked (not in Git repository), modified (not staged), or staged (need committed). Ex: $ git status -> would show the status of current branch and the changes made in it.
log: Displays commit history of the repository, with the option to format in various ways. Ex: $ git log
checkout: Used for switching or creating branches, and discarding changes for specific files. Ex: $ git checkout sprint01 -> would create and switch to the branch sprint01 if it didn't already exist, or it would just switch to it if it did exist.

3.3 Explain the difference between a branch and a tag.

Branches are used for active development starting from a specific commit that can be modified independent of the main branch, while a tag is a fixed reference to a specific commit. Tags are not used to actively develop anything, as you cannot commit or merge anything to them.

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

1. Easy collaboration. Being able to store copies of code on remote repositories and the ability to use branches allow people to easily work from geographically separated areas on the same project. You can view who made what changes and as a team decide what changes should be kept or need to be modified. In industry, large groups of people need to collaborate on projects without getting in each other's way. VCSs make it easier to manage.
2. Code backups. Since versions can be saved and accessed at any point from a remote repository, if local files got corrupted or a code idea didn't work out, you could revert to a previous working version.
3. Branching. Branching allows you to start from a specific point in your project history and actively work on it without potentially breaking or interfering with the main branch. In industry it would be especially helpful in situations where a project is actively deployed and needs to be

actively updated. Developers can work on branches until they have finished developing an update, at which point they can merge it with the main branch.

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.

1. What are you hoping to accomplish during this project?
2. How well do you handle conflict resolution and constructive criticism?

Resume of Kass Ramirez

Summary

I am currently a junior in advanced software engineering. I am hardworking and dedicated to learning new software engineering skills as well as new team skills. I would be a valuable asset to your team as I am the type to fully commit to my projects, and I am always willing to listen to others' input and advice.

Skills

· Conflict resolution

· Delegation

· Math

· Quick learner

· Listening

· Explaining concepts in digestible terms

· Open-minded

Relevant Experience

· Created the functionality of a bicycle race website in one of Deb’s other classes, where three-four iterations of the same project were done, adjusting to the changing desires of a customer over the course of development.

· Created a fully functioning RPG minigame in C# with graphics, stats, and an interactive display.

Interview Questions

1. What are you hoping to accomplish during this project?
2. How well do you handle conflict resolution and constructive criticism?
3. How do you handle suggestions for improvement when you feel confident in what you did? For example, if you designed and coded a method, and someone came and suggested some changes.
4. What role in a team do you wish to have?
5. What role do you think you’re currently best suited for with your current skill set?