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

Craig Lillemon

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**
Geeksforgeeks intro to virtual environments and python specifically - easy to understand and quick.	https://www.geeksforgeeks.org/python-virtual-environment/
A Google for Education full course walkthrough of python with lecture videos and text.	https://developers.google.com/edu/python/introduction
Broad but concise intro to version	https://homes.cs.washington.edu/~mernst/advice/version-contro

control and best practices. Also includes some git commands with explanations.	l.html
Longer walkthrough of git command line with decent in-depth explanations.	https://www.youtube.com/watch?v=e5wY8G00Ofl
Explains git and github, and the difference between the two	https://devmountain.com/blog/git-vs-github-whats-the-difference/#:~:text=Simply%20put%2C%20Git%20is%20a,help%20you%20better%20manage%20them.
Git command cheat sheet	https://education.github.com/git-cheat-sheet-education.pdf

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

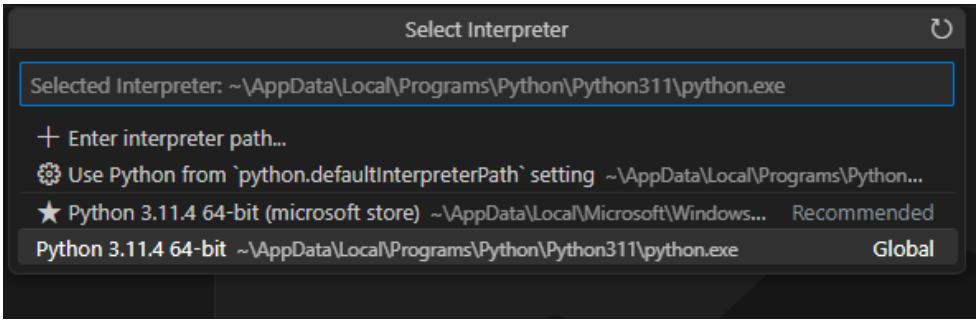
1. Open the command window and check your python version to see if you have it installed.
2. Install python version 3.11 Download Python. If on windows and have older version of python you should uninstall first : How to Uninstall Python

Install VS Code IDE

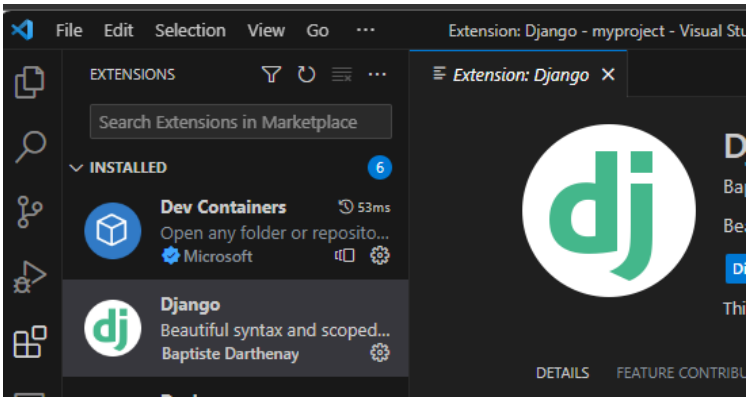
You can use a different IDE but this is what I will be using in my lectures. This has nice tools to integrate with python, django and databases.

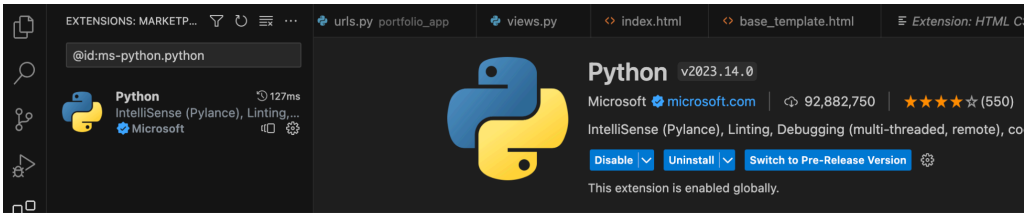
https://code.visualstudio.com/download

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`).

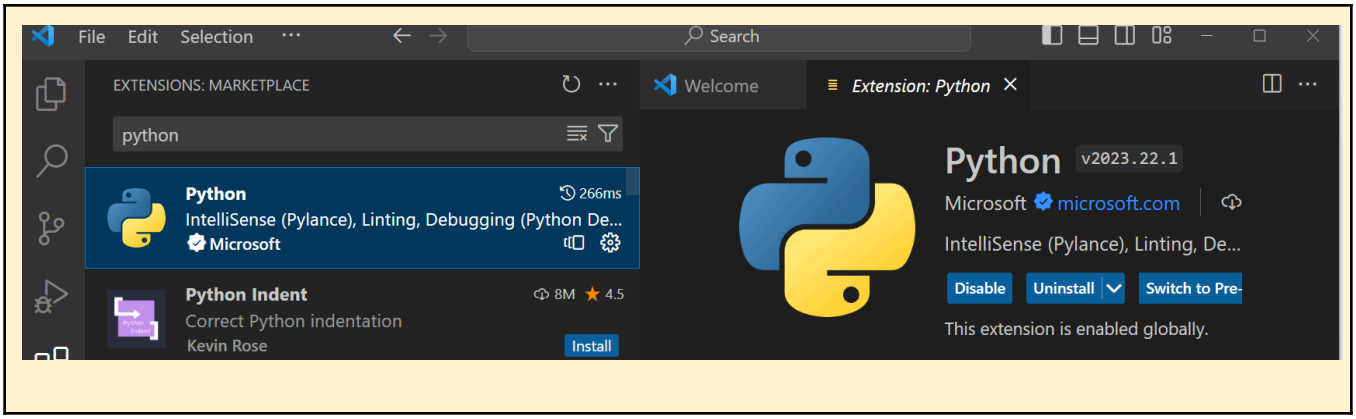


2. Install the Django extension developed by Baptiste Darthenay: In Visual Studio Code, go to the Extensions view and search for the "Django" extension. Install it to benefit from Django-specific features and enhancements for what we will be doing later.





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!

Variables and Types

Lists Review and complete 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")

# this code should write out the filled arrays and the second name in the names list (Eric).
print(numbers)
print(strings)
print("The second name on the names list is %s" % second_name)
```

Basic Operators Review and complete exercise code:

```
x = object()
y = object()

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

print("x_list contains %d objects" % len(x_list))
print("y_list contains %d objects" % len(y_list))
print("big_list contains %d objects" % len(big_list))

# testing code
if x_list.count(x) == 10 and y_list.count(y) == 10:
    print("Almost there...")
if big_list.count(x) == 10 and big_list.count(y) == 10:
    print("Great!")
```

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

Basic Operators

Math based, includes some lists

String Formatting

Similar to C % operator %s represents string and % represents integer

Basic String Operations

Ways to count string length, how many letters and so on

Conditions

Boolean operators, ==, less and greater, if and else, not

Loops

For loops, while, do while loops, break and continue

Functions Review and complete exercise code:

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

# 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:

```
# define the Vehicle class
class Vehicle:
    name = ""
    kind = "car"
    color = ""
    value = 100.00
    def description(self):
        desc_str = "%s is a %s %s worth $%.2f." % (self.name, self.color, self.kind, self.value)
        return desc_str
# your code goes here

car1 = Vehicle()
car1.name = "Fer"
car1.color = "red"
car1.value = 60000

car2 = Vehicle()
car2.name = "Jump"
car2.kind = "van"
car2.color = "blue"
car2.value = 10000


# test code
print(car1.description())
print(car2.description())
```

Dictionaries Review and complete exercise code:

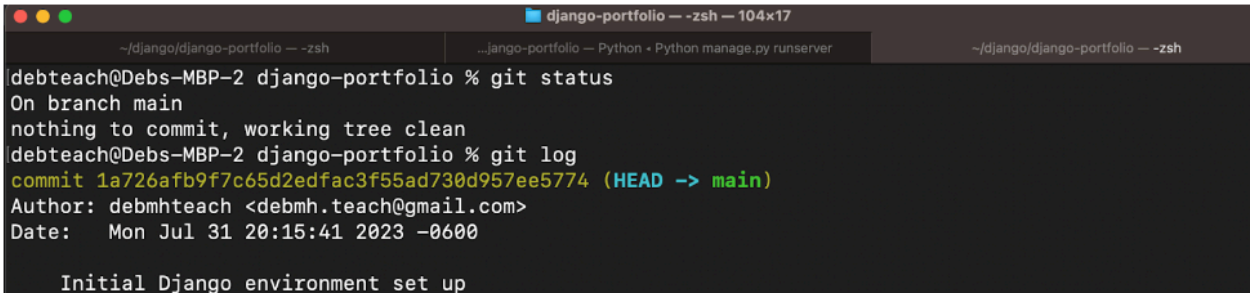
```
phonebook = {
    "John" : 938477566,
    "Jack" : 938377264,
    "Jill" : 947662781
}
# your code goes here
del phonebook["Jill"]
phonebook["Jack"] = 938273443
# testing code
if "Jake" in phonebook:
    print("Jake is listed in the phonebook.")

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



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.

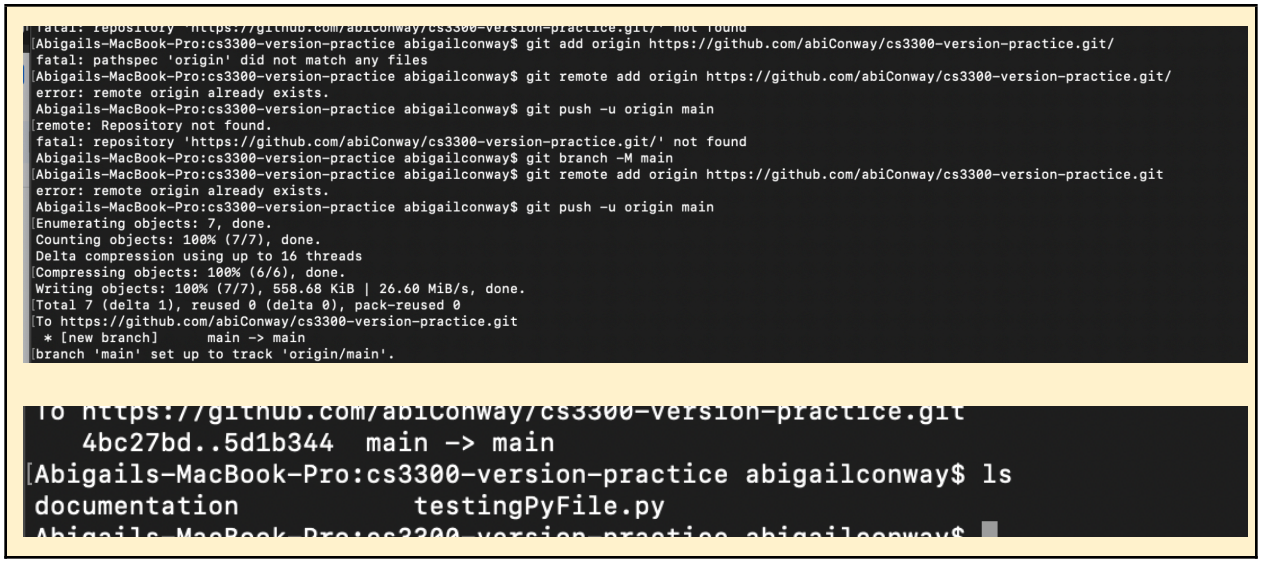
1. Create a python file in a local folder cs3300-version-practice
2. Create a folder called documentation in cs3300-version-practice that contains this document.
3. Create a github account if you do not have one.
4. Create a github repository that is public from the local folder.

Explain what you did and the commands you used.

cd cs3300-version-practice #change to local directory
git init -b main #initialize new git repository and set name of “-b” branch to “main” .git created in local folder
git add . #add all “.” changes to be staged for the commit
git commit -m “First commit” #commit all staged changes use “-m” to write commit message
On web browser: create repository under account on github.com, with “public” settings.
git remote add origin https://github.com/abiConway/cs3300-version-practice.git/ #add the remote destination to the local repository, Origin is just typical naming convention for remote repository

git branch -M main #rename the current branch to main. This was already done in step 2 using “git init -b main” but for some hiccup reason I needed to use this before pushing
git push -u origin main #Pushes committed changes to the remote repository. The “-u” sets the upstream to the main branch of origin, so you don’t have to specify that in the future.
*Side note: if you want to remove the local git repository use **rm -r .git**

Paste a screenshot of your local directory code



Paste a screenshot of your github repository code

Paste the url to you github repository code

https://github.com/abiConway/cs3300-version-practice/

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

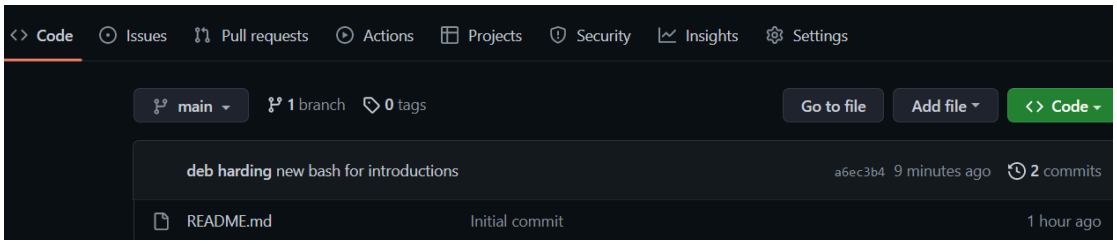
    2. Update the file. Before you can commit the version you must add the new file to the index (the staging area)

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

git log #check log
git branch #check what branch you are on
git branch sprint01 #create new branch by naming branch. ALTERNATIVELY: git checkout -b sprint01 would create new branch and move to it
git log
git branch

3. Switch to sprint01 branch to check out code:

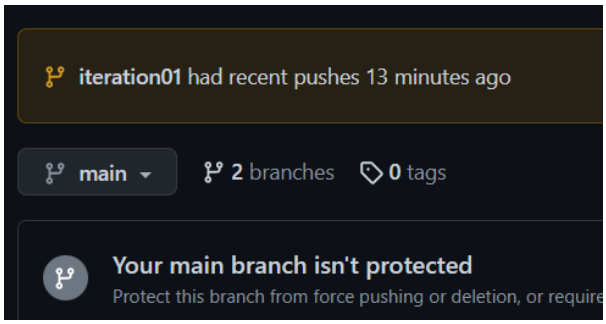
   git checkout 'sprint01'
   git branch
   git status

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

```
echo "#THIS IS AN EDIT" >> testingPyFile.py
git add testingPyFile.py
git commit -m "Modify python file"
```

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



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.

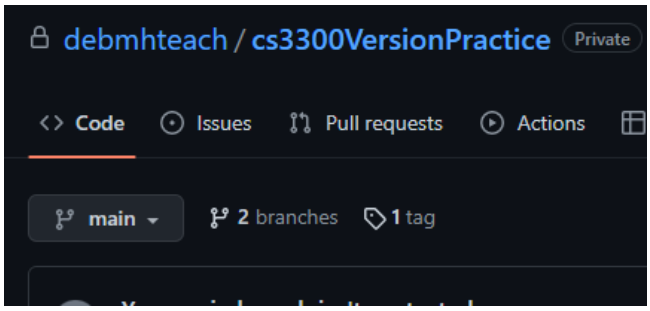
   Copy and paste the commands you used

```
git checkout main
git merge sprint01
git push
```

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 v1.0
git tag
git push origin v1.0
```

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 is a way for developers to work on, save, and update their code in a safe way. Basically, it’s a way to create a living history of your code in a time stamped filing system without the possibility of completely losing everything (mostly).

Each branch of a project is a new copy of the code it was created from. This new branch lives independent of other branches, even the one it was cloned from, until merged. A branch is a great place to keep a safe copy of working code, edit code, test code, share code, and more.

Commits are saving edits to the repository, in whatever branch is specified (from being in that branch or other). The new changes look seamless, but there is a record of the change.

Merging is updating branchA with the latest committed changes of branchB. Conflicts can arise if a branch tries to merge to another branch that's mutual base code has already been updated or changed.

A tag then marks important merges, a sort of timestamp for milestones in the code's development.

Version control allows you to mark the evolution of your code, keep old versions of your program, and keep your program functioning in its normal environment while you edit.

https://circleci.com/blog/git-tags-vs-branches/#:~:text=Branches%20allow%20you%20to%20code,together%20to%20optimize%20your%20workflow.

https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging

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

Git is a software, created in 2005, that is now the most popular version control software. It remains on your local machine. GitHub is a cloud based hosting service that holds git repositories for wider access. GitHub allows users to share their repositories, and let others edit code. Github is the remote, you need some sort of internet connection to get to it. Git is local and resides only on your machine.

https://devmountain.com/blog/git-vs-github-whats-the-difference

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

Commit saves your work from your local host to your local repository (ie to Git or a local directory).

Push saves your work from your local repository to your remote repository (ie on GitHub).

Pull brings updates from your remote repository to your local repository, where you can then edit the code.

add is just a way to let git know which files you would like to commit the next time the command is used - it doesn't actually affect any repository. But it puts the players you want on the stage for the commit.

Clone brings entire remote repositories to the local host.

Status will tell you exactly where your local version is in comparison to the remote repository, and if anything is waiting to be pushed or pulled.

Log shows the history of who committed what to the branch, it's just a log of changes.

Checkout allows you to switch between branches in the terminal, allowing for merging between branches.

https://education.github.com/git-cheat-sheet-education.pdf and others already mentioned in document

3.3 Explain the difference between a branch and a tag.

A branch is a space for your code to exist during ongoing development. A tag is like a chapter title in your code's history, it's a mark of a significant point in time for your code's development history, almost like a little birthday. Tags are usually used to mark a version of the code, Python 1, 1.2, 2, 2.2, ect. Branches are more of a workspace.

https://circleci.com/blog/git-tags-vs-branches/

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

If a program was being used, but needed to be updated. It would be costly and difficult to shut down the service that program provided while it was being edited. This way, programs can be updated and tested without affecting functionality except to merge the old version with the new one.

If an edit of a program completely broke the code, an earlier, functioning version of the program would be easily accessible, and very little work would have to be put into starting over.

Allowing multiple developers to work on the same project at the same time. Everyone has access to the base code, the updates, and the old edits, so that everyone is on the same page, individuals or groups can work independently and efficiency can increase.

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.