CSCI 3202 Introduction to Artificial Intelligence
Instructor: Hoenigman
Assignment 1
Due: September 4, by 4pm.

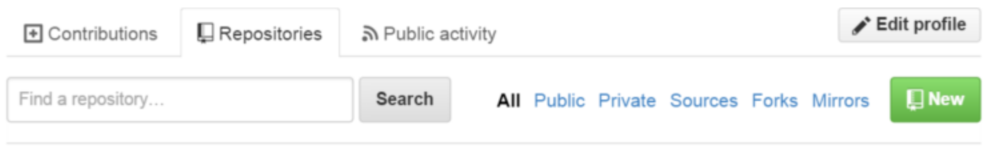
The purpose of this assignment is to introduce you to GitHub and python, if you are not already familiar with them, and provide a review a few essential data structures. You have two weeks for the assignment. I recommend starting early, especially if you're new to python or took data structures a long time ago.

Git to know GitHub

In this assignment, you will walk through the basics of GitHub by creating a GitHub account, creating a repository, and committing and pushing files.

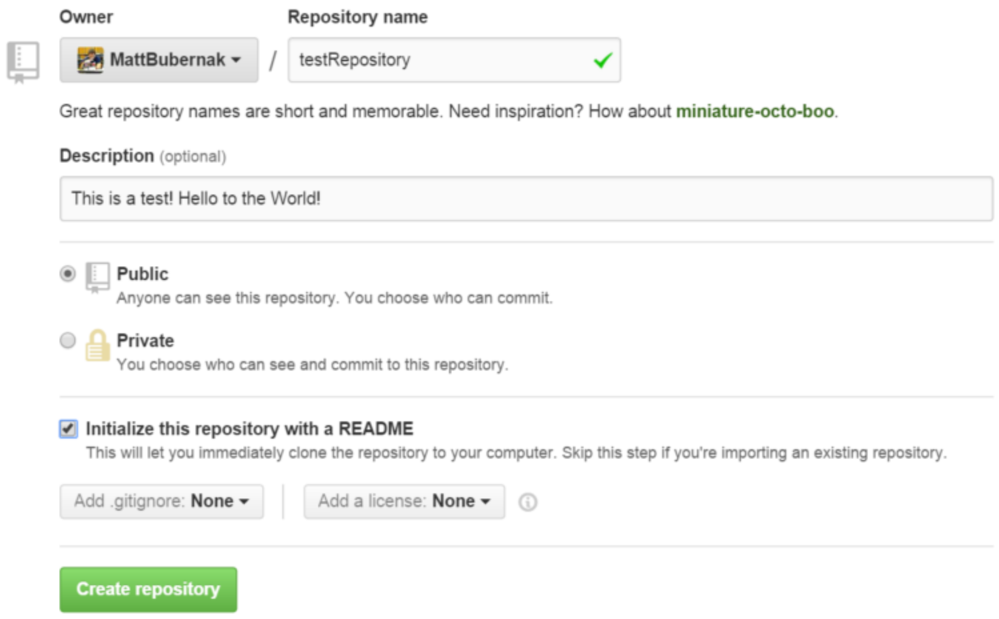
1. **Create a GitHub account**
    a. **If you already have a GitHub account, you don't need to create a new one. Just submit that username as Quiz 1 for this assignment.**
    b. Read up on the basics of GitHub at the following websites:
        i. https://training.github.com/kit/downloads/github-git-cheat-sheet.pdf
        ii. https://guides.github.com/
    c. Register for a GitHub account if you do not have one.
        i. https://github.com/join
        ii. Select "free" personal plan. You should use your "colorado.EDU" account to register. Student accounts get several added benefits.
    d. **Submit your username to the link for Quiz 1 on Moodle.**

2. **Create your own repository**
    a. Navigate to "repositories" tab from login page, select "new".



    b. Fill in a repository name, description, select the bubble for public, and check the box to initialize with a readme.

c. Your repository **must** be named: LastName_CSCI3202_Assignment1, where *LastName* is your actual last name.



Owner: MattBubernak / Repository name: testRepository

Great repository names are short and memorable. Need inspiration? How about miniature-octo-boo.

Description (optional): This is a test! Hello to the World!

(● Public) Anyone can see this repository. You choose who can commit.

Private — You choose who can see and commit to this repository.

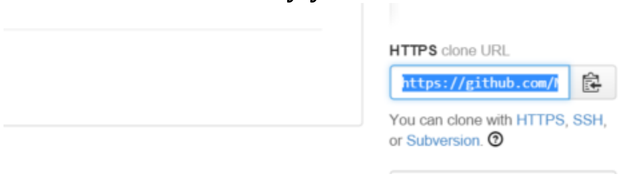
[✓] Initialize this repository with a README
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None | Add a license: None

3. Clone your repository

a. Test out your repository by “cloning” it to your computer.

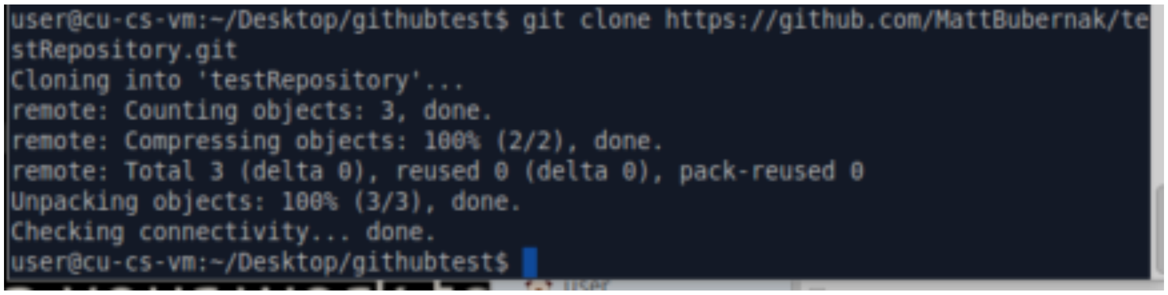
b. Identify your “clone url”, located on the page for the repository.



HTTPS clone URL: https://github.com/

You can clone with HTTPS, SSH, or Subversion.

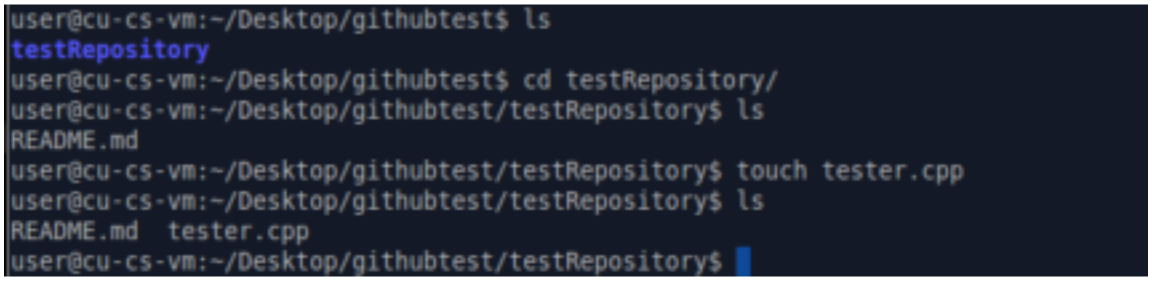
c. In a terminal, clone the repository to your machine: “git clone [clone-url]”



```
user@cu-cs-vm:~/Desktop/githubtest$ git clone https://github.com/MattBubernak/te
stRepository.git
Cloning into 'testRepository'...
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
Checking connectivity... done.
user@cu-cs-vm:~/Desktop/githubtest$
```

4. Add a file to the repository.

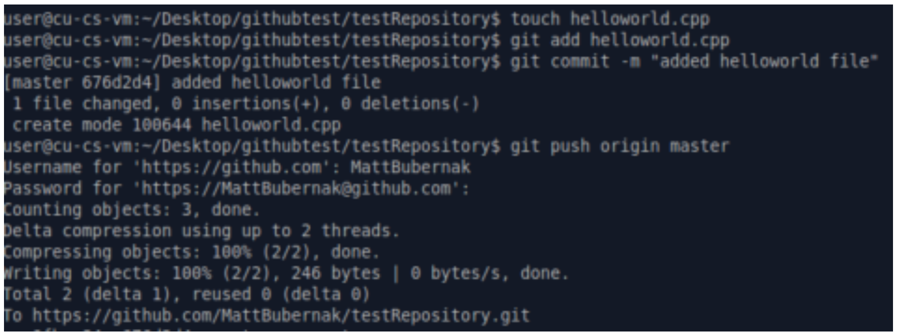
a. Create a file inside of the local clone of your repository called *LastName_Assignment1.py*. You can create the file in your favorite text editor or IDE.



```
user@cu-cs-vm:~/Desktop/githubtest$ ls
testRepository
user@cu-cs-vm:~/Desktop/githubtest$ cd testRepository/
user@cu-cs-vm:~/Desktop/githubtest/testRepository$ ls
README.md
user@cu-cs-vm:~/Desktop/githubtest/testRepository$ touch tester.cpp
user@cu-cs-vm:~/Desktop/githubtest/testRepository$ ls
README.md  tester.cpp
user@cu-cs-vm:~/Desktop/githubtest/testRepository$
```

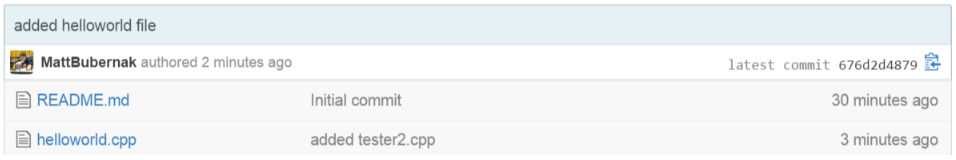
   b. Add this file to the repo using “git add [filename]”, which means it will be included in your next commit.

5. **Commit and push your changes**
   a. Commit your changes with a message: “git commit -m “[message text]””
   b. Push your changes to your repository “git push origin master”



```
user@cu-cs-vm:~/Desktop/githubtest/testRepository$ touch helloworld.cpp
user@cu-cs-vm:~/Desktop/githubtest/testRepository$ git add helloworld.cpp
user@cu-cs-vm:~/Desktop/githubtest/testRepository$ git commit -m "added helloworld file"
[master 676d2d4] added helloworld file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 helloworld.cpp
user@cu-cs-vm:~/Desktop/githubtest/testRepository$ git push origin master
Username for 'https://github.com': MattBubernak
Password for 'https://MattBubernak@github.com':
Counting objects: 3, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 246 bytes | 0 bytes/s, done.
Total 2 (delta 1), reused 0 (delta 0)
To https://github.com/MattBubernak/testRepository.git
```

Good Work! Now you should see your changes reflected on your repository page.



added helloworld file

MattBubernak authored 2 minutes ago | latest commit 676d2d4879

README.md	Initial commit	30 minutes ago
helloworld.cpp	added tester2.cpp	3 minutes ago

Python tutorial and data structures review

Once you’ve created an Assignment 1 repo and python file, it’s time to jump into python and a review of the data structures you will encounter this semester. There is a tutorial at Learn Python the Hard Way: http://learnpythonthehardway.org/book/ for your reference. If you’ve never used python, you should work through the exercises in the book to familiarize yourself with the language.

The first four programming questions are implementations of data structures you have hopefully seen previously in other languages. The fifth question is to write the test to demonstrate the functionality of your data structures.

Assignment 1 Programming questions

1. Implement a **queue** using the python queue module. Your queue should handle integers only.
2. Implement a **stack** using a list to store the stack data. The stack should be implemented as a class with at least three methods:
    a. *push(integer)*
    b. *pop()*
    c. *checkSize()*
3. Implement a **binary tree** class. The tree should be made up of nodes, which are also implemented as a class. The tree should have a root node. Each node in the tree has four properties:
    a. *integer key*
    b. *left child*
    c. *right child*
    d. *parent.*

    Your tree class should have methods to add a node, delete a node, and print the key values of the nodes. The specific add, delete, and print functionality is as follows:
    e. *add(value, parentValue)*
        i. *value* is the key value for the new node and *parentValue* is the key value of the parent. If the *parentValue* is found in the tree, then:
            1. add the new node as the left child if the parent has no children.
            2. add the new node as the right child if the parent has a left child only.
            3. Don't add the node if the parent already has two children. Print a message, "Parent has two children, node not added".
            4. if *parentValue* is not found in the tree, then print a message, "Parent not found".
    f. *delete(value)*
        i. For the delete operation, you only need to delete nodes that don't have any children.
            1. If a node has 1 or 2 children, print a message, "Node not deleted, has children".
            2. If the node is not found, print a message, "Node not found."
    g. *print()*
        i. For the print operation, print the parent key value and then the key values of the children. This is a pre-order traversal.
4. Implement a **graph** class for an unweighted graph using a dictionary to store the graph data. Each vertex in the graph has an integer key value and a list of adjacent vertices. Your class should include the following methods:
    a. *addVertex(value)*

        i. This method first checks that the vertex value doesn't already exist in the graph, and then adds it to the graph. If the vertex is already in the graph, print a message, "Vertex already exists".

    b. *addEdge(value1, value2)*

        i. This method takes the key values of two vertices in the graph and adds an edge between them. If one or both of the vertices don't exist in the graph, the method should print a message, "One or more vertices not found."

    c. *findVertex(value)*

        i. This method takes the key value of the vertex to search for, and if it's found in the graph, print the key values of its adjacent vertices.

5. Write the code to test your data structures in your *LastName_Assignment1.py file*. The test code should clearly show which data structure is being tested the methods being called, the arguments to the method, and the state of the data structure after the operation. Your test code should also be well documented so that the grader can easily see what you've done.
    a. Testing the queue:
        i. Add at least 10 integers to the queue and then dequeue them and print their values in the order they are dequeued.
    b. Testing the stack:
        i. Add at least 10 integers to the stack and then pop the values and print their values in the order they are popped.
    c. Testing the tree:
        i. Add at least 10 integers as nodes to the tree.
        ii. Print the tree.
        iii. Delete at least 2 integers from the tree.
        iv. Print the tree.
    d. Testing the graph:
        i. Add at least 10 integers as vertices to the graph.
        ii. Add at least 20 edges to the graph.
        iii. Find 5 vertices in the graph.

What to Submit and other important information:

1. Submit your *username* to Moodle for Quiz 1.

2. For this assignment, you need to submit your python code to the Assignment 1 link on Moodle. For future assignments, we intend to grade it right on GitHub.

3. Your GitHub repository needs to be named *Lastname_CSCI3202_Assignment1* for you to get credit.

4. Your repository needs to have at least one file named *Lastname_Assignment1.py* for you to get credit. You can have additional *.py* files in your repository.