Python Lab Setup for RIT Courses

Tools get updated, and sometimes procedures change. If any information below seems incorrect or somehow out of date, please tell an instructor.

Here you will find complete instructions for setting things up and creating and submitting a first "toy" lab. Many of these instructions describe what you must do every time you work on a new assignment. To help you get an idea of what gets repeated and what does not get repeated, the steps that you in some way will repeat for each assignment are marked with a triangle (△).

Python and PyCharm Setup

Go to http://www.python.org and download/install the latest version 3 release version (e.g. 3.7.x).

> (Some computers come with Python version 2.x installed. This is not compatible with the materials in our courses.)

1. Go to https://www.jetbrains.com/pycharm/ and download/install the latest version. The free Community Edition is all that is required for this course. If you download the Professional Edition, you will have to register for a free 1 year license with Jetbrains using your university email account.

2. Launch PyCharm. In the splash screen, select **Configure** → **Preferences** (MacOS), **Settings** (Windows). Under **Project Interpreter**, if the interpreter is not found you will have to navigate to it. Click on **...** button to right and select **Add Local**. Navigate from the root folder to the location of the interpreter on your machine, e.g.:

MacOS
    /Library/Frameworks/Python.framework/Versions/3.7/bin/python3
Windows
    C:\Users\username\AppData\Local\Programs\Python\Python37\python.exe

Select the **OK** button.

3. Create a place on your personal computer to store your programming assignments. For example, in your **Documents** folder, create a top level **CSAPX** folder, with two subfolders, **Labs** and **Projects**.

4. △ In the PyCharm splash screen, select **Create New Project**. For **Location**, change to your Labs subfolder and then append a new subdirectory **Lab0** i.e. **...Documents/CSAPX/Labs/Lab0**. Make sure that **Pure Python** is selected to the left, and **Interpreter** is set correctly. Select the **Create** button.

5. △ In the Project window (**View** → **Tools Window** → **Project**), right click on the **Lab0** folder and select **New** → **Directory** and name the directory **src**. Select **OK**. Now, right click on the **src** directory and select **Mark Directory As** → **Sources Root**. The directory color should change to blue. This is where you will put your source code. By marking it as sources root, PyCharm will automatically be able to resolve all packages used.

6. Right click on the **src** directory and select **New** → **Python File**. Name the file **square.py**. In the text editor, copy/paste the following program from: https://www.cs.rit.edu/~csapx/Resources/square.py

7. To run the program for the first time, right click in the text editor and select **Run 'square'**. The console should display a message in a window below asking to Enter side length: . In the console, enter the value 100. It should proceed to open a new window that draws a square of 100 units in length. The console will then indicate that you can

end the program by closing the drawing window. Close the drawing window and you should see that the program terminated in the console.

8. Now we will feed the side length of the square as an argument on the command line. Find the green run icon in the upper portion of the main window. To the left, left click on **square** and select **Edit Configurations**. Under script parameters, enter **200**. Select the **OK** button and then run the program again using the green run icon. It should automatically draw the square of side length 200, without prompting.

Git Setup

1. MacOS users need to verify you have git on your machine. Run the terminal application **Go** → **Applications** → **Utilities** → **Terminal.app** and type in:

```
which git
```

If you get a path back, you probably got the program installed with the Xcode application. If you have never run **git** before, run it:

```
git
```

At that point you will be given an end user license agreement (EULA) that you must accept.

If you have run **git** before, you will just be given some tips on options. You can skip the next two steps. If you don't have Xcode installed and have never otherwise installed **git**, you can choose to install the Xcode application or just follow the Windows instructions in the next two steps.

2. Windows users must first get git and install it on your machine. Go to https://git-scm.com/download/ and run through the installation. Make note of the path where Windows installs the Git application.

3. In the terminal window (Mac/Unix) or Git bash shell (Windows), set up your desired username and email using the two commands:

```
git config --global user.name "John Doe"
git config --global user.email johndoe@example.com
```

4. In the PyCharm project window, go to **PyCharm** → **Preferences** (MacOS), **Settings** → **Version Control** → **git**. Enter the path under **Path to git executable** and then select the **Test** button to verify it is successful.

5. In the PyCharm project window, select **VCS** → **Import into Version Control** → **Create Git Repository…**. Select the **Lab0** folder to create the new repository into and select the **OK** button. You should notice that the displayed name for the file named **square.py** has changed to red. This means it is not being tracked for version control.

**** IMPORTANT **** If a dialog ever comes up asking if it should automatically add files to git, you should normally only allow it if it is a source file or something else you wrote. PyCharm will create several machine-specific files that should not be put under source control. You can manually add the **src** folder to git simply by right clicking on the **src** folder and selecting **Git** → **Add**. When this is done, you should see that **square.py** is now green. The green color means it is a new file under version control that has not been checked in yet.

> As your sophistication level increases, you may find that saving certain configuration files can actually be helpful as you develop code on several different personal computers. A rule of thumb is to never put any file under **git** control if that file mentions any absolute paths that only make sense on your current computer.

You cannot proceed past this point until you get your CS account information.

---

5. Now that your local machine git repository is set up, you will log into the CS machines and create a remote "master" repository that you can connect to. This remote repository will serve as the central location and back up space. You must connect to the CS machines using a text-based, terminal window and the *secure shell* (SSH) protocol.

For MacOS users, launch the terminal application, **Go** → **Applications** → **Utilities** → **Terminal.app**. For Windows users, you can download and run MobaXterm, http://mobaxterm.mobatek.net/, and select **Start local terminal server**. In the command shell enter **ssh *username*@glados.cs.rit.edu** (replace *username* with your actual CS user name). Enter your password and you will be connected to your home directory on the CS machines.

> **glados** is a high-volume Linux computer in our computer room. Since all of our Linux computers mount your home file system, you can choose to use a different host if you wish. Another machine in the computer room is **queeg**. In reality, you could use any Linux computer you see in our labs as well.

6. First, you will create the remote repository. It will reside under the top level directory **Git**. Since you will likely be using **Git** for other courses, we will create a directory that is specific to this course, as well as subdirectories for labs and projects. Run the following commands:

```
mkdir -p ~/Git/CSAPX/Labs
mkdir -p ~/Git/CSAPX/Projects

git init --bare ~/Git/CSAPX/Labs/Lab0
```

Note: Linux systems are case-sensitive, but Windows systems are NOT.

7. Next, we want to get the fully qualified path to the new repository you made for **Lab0**. Change to the directory and copy the path returned by the **pwd** command:

```
cd ~/Git/CSAPX/Labs/Lab0
pwd
```

Do not exit the terminal window! You will need to come back to it later on in this tutorial.

8. Now we can go back to your local machine running PyCharm. We are going to commit the changes you made to your local repository, as well as connect and push the changes to the remote one.

Right click on the **Lab0** folder in the project window and select **Git** → **Commit Directory…**.

> You can also click on the little green upward-pointing arrow in the tool bar.

In the commit dialog, enter a commit message, e.g. "Initial revision". Then select the **Commit** menu button and carefully make sure to select **Commit and Push…**.

In the push dialog, click on **Define remote** to the right of **master**. You will now specify the full path to your remote repository in the new dialog. For URL enter **ssh://*username*@glados.cs.rit.edu/*path***. Here, *username* is your CS username, and *path* is the path you copied from the **pwd** command earlier. For example, if my user name was **xyz9999** and the path was **/home/stu3/s12/xyz9999/Git/CSAPX/Labs/Lab0**, the URL should be **ssh://xyz9999@glados.cs.rit.edu/home/stu3/s12/xyz9999/Git/CSAPX/Labs/Lab0**.

Select the **OK** button and agree to connect to the remote machine (enter CS password when prompted). If the connection is successful, you will see the file you are about to push out, along with the commit message. Select the

Push button to push the change to the remote machine.

After connecting again, you should see a message saying the push was successful. In the project window, you should see that the name **square.py** is now black, which means the file is checked into version control and has not been modified since the last commit.

Try Setup

1. Back in the terminal window connected to the CS machine, register your account with **try**.

```
try csapx-grd register /dev/null
```

Enter the course and section number that you are registered in. If you make a mistake, you need to let your instructor know immediately.

2. We will now make a "CS Account" checkout *clone* of the code you committed to the bare repository so that you can submit it to try. The bare repository you made and to which you checked in your code earlier is not meant to be an area where you view, edit, or run code. Think of it as a compressed version of your code. In order to view it, you need to *clone* it into a new directory. We will first create a common place for you to clone the code from the bare repo:

```
mkdir -p ~/Courses/CSAPX/Labs
mkdir -p ~/Courses/CSAPX/Projects
```

Now change to the lab directory and issue a clone:

```
cd ~/Courses/CSAPX/Labs
git clone ~/Git/CSAPX/Labs/Lab0
```

You can now change into the **Lab0/src** directory to see the latest version of **square.py** that was committed.

```
cd Lab0/src
ls -l square.py
cat square.py
```

3. You can now submit this file to the try target:

```
try csapx-grd lab0-1 square.py
```

If try is successful, you will see that the file was submitted. If there are errors, it will not save. You can submit to try as many times as you want, but it will only save the last successful submission. To verify what was submitted, you can use the **-q** option:

```
try -q csapx-grd lab0-1
```

4. Now let's go back to your personal machine and the PyCharm project. Change **square.py** so that it draws two squares, side by side. Run it to make sure it is correct. Notice how the file has changed to green, meaning there have been changes since the last commit. Don't forget to add yourself as author in the file comment block. Right click on **Lab0** and select **Git → Commit Directory**. For the commit message, put "Changed program to draw two squares". Then select the **Commit and Push** button. Go through the login credentials and you should see the file being committed locally, as well as pushed out to the remote repository.

5. △ Back in your CS Account terminal window, you are in the cloned directory (you can always run the **pwd** command to see where you currently are). If you look at **square.py**, you will see that it hasn't changed. In order to get the change that you committed to the bare repo, you must **pull** it into the cloned area:

```
git pull
```

Now you will see the update. In this course, you will always be required to submit your git log history. To generate this at any time you can run the command:

```
git log > log.txt
cat log.txt
```

It will create a CS account file, **log.txt** with the full commit history. You can submit this along with your code to the same target:

```
try csapx-grd lab0-1 square.py log.txt
```

Again, run the verify command to see that both files are present:

```
try -q csapx-grd lab0-1
```

All Set!

When you come to the first class, make sure you ask questions if you have any. You will be repeating some of this every time you work on a lab or project.

Sun Jan 14 12:47:35 EST 2018