**Lab 1**

**1 Installation and Setup of Git Command Line**

**1.1 Download the Git installer from the link below:**

<https://git-scm.com/download/win>

Change:

Default Editor: Notepad

Terminal Emulator to use with Git Bash: Use Windows’ default console window.

**1.2 Open a Command Prompt (cmd) window.**

**1.3 At the command line, type the following command to set up a new Git user.**

*git config --global user.name <your name>*

Example: *git config --global user.name "KweeTeck "*

**1.4 To link an email address to the newly created Git user, type the following command.**

*git config --global user.email <your email>*

Example: *git config --global user.email tan\_kwee\_teck@ichat.sp.edu.sg*

**1.5 After you have created a new Git user and linked an email address to this user,**

**verify that Git has saved the new user settings using the command below.**

*git config --list*

A screenshot of a computer

Description automatically generated with medium confidence

The newly created username and email are not used to authenticate with a remote Git server or services such as GitHub, Gitlab, etc.

Instead, Git will append the current user information with username and email address to each Git “Commit” comment whenever new or modified files are “Committed” to a Git local repository

**2 Creation of a new local Git Repository**

**2.1 The first step to using Git is to create a new folder to keep files in this directory to be added to the Git repository. Create a new folder on your laptop’s hard disk.**

For example, if you store your data in C-drive of your laptop, you may create a new folder “Local\_Git\_Repository” and the directory of this new folder will be C:\Local\_Git\_Repository.

**2.2 To create a new Git repository, first change to the directory where you want to create a new Git repository. You may set to a path in your C-drive or other drives in your laptop.**

*cd <new folder’s path>*

Example: *cd “C:\Local\_Git\_Repository” or cd C:\Local\_Git\_Repository*

**2.3 Once you have changed to the directory where you want to create a new directory for your new Git repository, run the DOS command below**

*mkdir <name of new directory for new Git repository>*

Example: *mkdir lab1* A picture containing text, font, screenshot, white

Description automatically generated

**2.4 After changing to the directory of the new repository, run the following command.**

*git init*

The "git init" command is used to initialize a new Git repository. It creates a .git subdirectory that contains all the necessary files for Git to operate. It's important to note that "git init" only needs to be run once per repository. If you've already initialized a repository in a directory, running "git init" again will overwrite your existing repository with a new, empty one.

If the new Git repository has been successfully initialised, you should see the following status in the command prompt:

A picture containing text, font, receipt, white

Description automatically generated

**2.5 Using Windows Explorer, open the Local\_Git\_Repository folder. You should see that a new folder “.git” is created, as shown below.**

Take note that “.git” is a “hidden item”, therefore please make sure that you configure your File Explorer to show “hidden items”. (Procedure: In File Explorer, click “View” and put a tick to “Hidden items”. You should also tick the “File name extensions” option, so that the full file name of your files will be shown.)

This new “.git” directory contains the Metadata information that tracks the files contained in the Git repository. It maintains the different versions of each file in the repository and tracks all commits, branches, etc that are done on this repository.

**3 Committing Files to a local Git Repository**

After creating a new Git repository, the next step is to add files to this repository.

“Committing” a file to a Git repository means that Git will track the status of this file, if it has been modified, all the different versions of this file that have been previously committed, and the developers that have committed the specific file versions.

**3.1 Using any text editor (e.g., Notepad), create a new Python file “HelloWorld.py”**

**with the following Python code in the root directory of your new Git repository.**

A screenshot of a computer error

Description automatically generated with low confidence

**3.2 Run the git command below to check the status of the Git repository.**

*git status*

After running this command, you should see that the new “HelloWorld.py” file is in the Git repository root directory, but “HelloWorld.py” is listed under “Untracked files:”. This is because although the file “HelloWorld.py” is physically located in the directory of the Git repository, we have not yet committed this file to the Git repository.

A screenshot of a computer

Description automatically generated with medium confidence

**3.3 Before committing a new or modified file (in this case, HelloWorld.py) to the Git repository, you need to add this file to the “staging area” using the following command.**

*git add HelloWorld.py*

“Staging files in Git” is the processing of adding snapshots of the files to the staging area, preparing them for the “Commit” action. Take note that “git add” command does not commit the file to the Git repository.

**3.4 After adding the new file “HelloWorld.py” to the staging area, run the git status command to verify that the file has been added to the “Staging Area”.**

*git status*

A picture containing text, screenshot, font, line

Description automatically generated

**3.5 Now that “HelloWorld.py” has been added to the staging area, we need to “commit” this file to the Git repository with a short text comment that describes the changes made in this commit.**

*git commit -m "Initial version to display text message on console"*

A close-up of a message

Description automatically generated with low confidence

**3.6 After the “commit” is completed successfully, run git status again to verify that all files in the current base directory are committed successfully to the Git repository.**

*git status*

A picture containing text, font, screenshot, line

Description automatically generated

If any existing files in the Git repository is modified, you can use command git status to see if any files have been modified and not yet committed.

**3.7 Comparison of Changes in Files**

Let’s now modify the HelloWorld.py file and make use of git command to find the differences. Open the HelloWorld.py and append the text “ is a DCPE module”) to it. Remember to save file

**3.8 Since you made some changes to the already-committed file “HelloWorld.py”, then this should be reflected when you run the command *git status* again.**

A screenshot of a computer error

Description automatically generated with low confidence

**3.9 You can use the git command *git diff* to view the file differences before deciding to either commit the file again to the Git repository or continue modifying the file and commit the changes to the Git repository later.**

A screenshot of a computer program

Description automatically generated with medium confidence

**4 Branches in Git**

**4.1 To view all the branches created in the current Git repository, use the command git branch without any parameters.**

*git branch*

This command lists all the available Git branches in the current repository and the current selected branch is indicated with a “\*” asterisk prefix and highlighted in green color below. In the example shown below, “master” is the selected branch.

A picture containing text, screenshot, font, line

Description automatically generated

**Create new Git branches**

You can create new branches in Git repository. Before creating a new branch, first run the git branch command to view all the available branches and check that you are in the correct branch that you would like to create a new branch from.

To create a new branch, use the git branch command with a parameter specifying the name of the new branch:

*git branch <new branch name>*

**4.2 Using the “master” branch as the base branch, create a new branch “bug-fix1” using the command below.**

*git branch "bug-fix1"*

**4.3 After creating the new “bug-fix1” branch, use the git branch command to verify that the new branch creation is successful.**

**Checkout (switch) branches**

To switch to a different Git local branch, you need to “checkout” that branch, using the git checkout command:

*git checkout <branch name>*

**4.4 Now, try switching to the new branch using the git checkout command. Then use the git branch command to confirm that you have switched to the new branch.**

**4.5 You should see that the “\*” is now placed at the line of “bug-fix1”.**

*git branch*

*git checkout "bug-fix1"*

*git branch*

A screenshot of a computer program

Description automatically generated with medium confidence

**Merging Git Branches**

Git branches can be used for several different reasons, for dedicated bug fixes, new features, software releases, etc.

However, sometimes we will also need to merge the branches into new or existing branches if We need the code changes implemented in one branch in another branch.

To merge a branch into the current branch, use the command:

*git merge <branch to be merged>*

**4.6 You will now try merging branches. First, check again that you have checked out the “bug-fix1” branch.**

**4.7 Next, make the small code change in the file HelloWorld.py as shown below.**

A screenshot of a computer

Description automatically generated with medium confidence

**4.8 Commit the changes to the new branch “bug-fix1”.**

*git add .*

*git commit -m “Merge”*

**4.9 Now, you are going to merge this change done in the “bug-fix1” branch into the “master” branch. You need to switch back to the “master” branch first.**

*git checkout "master"*

**4.10 Merge the branch “bug-fix1” into the current “master” branch using the git merge command.**

*git merge "bug-fix1"*

**4.11 Run the git branch command to list the branches in the Git repository. Notice that the branch “bug-fix1” remains (does not disappear) even after the branch have been merged.**

**5 Creating a GitHub account**

GitHub is a popular cloud-based remote Git repository where developers can upload their software projects for collaboration and sharing with other developers.

In Git, you can use GitHub to “push” your code to GitHub where your code is uploaded to the cloud and can be shared with others.

**5.1 First, you will create a free GitHub account. Using the link below, sign up for a free GitHub account.**

**5.2 Register the new GitHub account using your SP iChat email account.**

<https://www.github.com/>

**Create a new GitHub Repository**

GitHub organizes code projects based on repositories, so before you can push your code from your local Git repository to GitHub, you first need to create a repository in GitHub.

**5.3 After logging into GitHub in the main page, click +∇ and select “New Repository” to create a new GitHub repository.**

A screenshot of a computer

Description automatically generated with medium confidence

**5.4 Fill in the field for the “Repository name” (use Lab 1) and set the repository to “Public” (radio button). Click the “Create Repository” button at the bottom of the page to continue.**

A screenshot of a computer

Description automatically generated with medium confidence

**Push contents from Local Git Repository to Remote GitHub repository**

After creating the remote GitHub repository, you can “push” (upload) your Local Git Repository to the new remote GitHub repository.

**5.5 In the GitHub repository, under the “Quick setup…”section (see Figure 17), click the copy button to copy the URL link of the repository. The link should be something like this:**

https://GitHub.com/<your name>/Lab1.git

A screenshot of a computer

Description automatically generated with low confidence

Before pushing your Local Git Repository to GitHub, you need to first specify the URL of the remote GitHub repository by executing the git remote command below.

*git remote add origin <GitHub repository URL from Step5.5>*

Example: *git remote add origin https://GitHub.com/kweetecktanichat/Lab1.git*

**5.6 To check if the currently local Git repository has the remote repository server correctly configured, run the following Git command below**

*git remote –v*

**5.7 After configuring the URL for the remote GitHub repository, you now need to setup the remote upstream branch (the ‘master’ branch in your GitHub repository) before you can push your local commit changes to GitHub. To do so, use the git push command below.**

*git push --set-upstream origin master*

**5.8 Finally, you push the Local Git Repository to your remote GitHub repository using the git push command below.**

*git push –u origin*

The git push command should be successfully completed as shown in the command window below.

A picture containing text, screenshot, font, number

Description automatically generated

**6 GitHub Readme file**

For each GitHub repository, it’s usually a good practice to create a Readme file which then appears in GitHub as the main webpage for a repository.

The GitHub Readme file syntax is based on a custom Markdown syntax which is documented in the link below.

<https://GitHub.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet>

In this part, you will create a text file “Readme.md” and push it to your remote GitHub repository.

**6.1 Using Notepad, create a new text file “Readme.md” in your D:\Local\_Git\_Repository directory.**

Remember to name the file as Readme.md, not Readme.md.txt. You will be warned that changing the filename extension from might make the file unusable. Ignore the warning.

**6.2 In the Readme.md file, add a H1 header with the text “ET0735 – Lab 1 (Introduction to Git and GitHub)” (See Figure 22).**

Make sure that there is a space between “#” and “ET0735 – Lab 1 (Intro….)”. Save the file and close the text editor.

A screenshot of a computer program

Description automatically generated with medium confidence

**6.3 Commit and push the new Readme.md file to GitHub.**

**6.4 Reload the URL for your GitHub repository page and check that your Readme.md file is now the main HTML page for your Lab1 GitHub repository.**

Explore further syntax for readme.md in the following link

<https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-andformatting-on-github/basic-writing-and-formatting-syntax>

Experiment with various syntax codes such as changing the font style, adding links and images to your project.

**7 Creating a Git tag and GitHub Release**

Tags are used in Git usually to create Software Releases. A Software Release is basically a collection of files with each file having fixed version.

In Git, we use “Tags” to mark these collection of files in a specific branch and these “Tags” are basically a text string such as “v1.0”, “v1.1”, etc which typically describes the Software Release number.

To create a tag in Git, first check that you are in the correct branch, else switch to the branch (git checkout) you want to create the tag on. Once you are in the correct branch, use the *git tag –a <tag> –m <comment>* command to create a new Git tag on the current file versions in this branch.

Example: *git tag –a v1.0 –m "Initial release v1.0"*

Now, you are going to create a new tag, and apply the new tag to your Local Git Repository. You will then push the tag to your GitHub repository.

**7.1 Before creating a new tag, verify that your GitHub repository shows that it has no tag (i.e. 0 tags).**

**7.2 Check that you are at the “master” branch of your Local Git Repository.**

**7.3 Create a new git tag “v1.0” using the git tag command below.**

*git tag –a v1.0 –m "Initial release v1.0"*

**7.4 You will now push the newly created tag from the local git repository to the remote GitHub repository. The git push command used so far for pushing files needs to be modified slightly to push git tags. The modified git push command is shown below.**

*git push origin v1.0*

A screenshot of a computer code

Description automatically generated with low confidence

**7.5 Reload the URL for your GitHub repository page and observe that now it shows that there is “1 tag”.**

**7.6 Next, you are going to create a software release v1.0. Click the “1 tag” icon at your GitHub repository, then click “Releases”.**

The page will show “There aren’t any releases here” since you have not created any release yet.

**7.7 Click the “Create a new release” button. Click “Choose a tag” and select “v1.0”.**

In the “Release title” field, enter “ET0735\_Lab1\_Code\_v1.0” (or whatever text you prefer). In the “Describe this release” field, enter two lines of text: “Initial release v1.0”, and “Date: {today’s date}”.

A screenshot of a computer

Description automatically generated with medium confidence

**7.8 Click the “Publish release” button at the bottom.**

**7.9 Go back to the main page of GitHub repository. You should see a new release.**

A screenshot of a computer

Description automatically generated with medium confidence

**8 Git Submodules**

Until now we have learned how to create a single Git repository which you can use to add, commit, and push your files to Git and GitHub.

However, when working on larger projects such as the Mini-Project it’s sometimes necessary to reference external shared Git/GitHub repositories that are maintained centrally by another team but used in the repositories of other teams.

**8.1 Creating Git submodules**

To create a Git submodule you first need an existing local Git repository to add the Git submodule into. Git submodules are basically existing GitHub repositories which are then referenced by your local Git repository.

The Git command syntax is shown below and should be executed at the root level of an existing repository.

*git submodule add <Git submodule repository URL>*

Example: *cd C:\Local\_Git\_Repository*

*git submodule add https://github.com/JohnProject.git*

**8.2 Creating Git submodules from existing GitHub repositories**

In this exercise, we will create a new Git submodule by adding an existing GitHub remote repository.

Change to the root directory of the Git repository you’ve created earlier for Lab 1.

Using the command below, we will add a new Git submodule from an existing GitHub repository

*git submodule add https://github.com/ET0735-DevOps-AIoT/Lab1\_submodule.git*

**8.2.1 Additional Git files created for Git submodules**

After you’ve created and added the Git submodules in the previous step, observe that Git has automatically created a new hidden file “.gitmodules” in the root directory of your Lab 1 local repository.

A screenshot of a computer

Description automatically generated with medium confidence

.gitmodules is basically a text file which lists the local directory path of the submodule and also its corresponding remote URL. Open the .gitmodules file with Notepad to observe its content. A screenshot of a computer code

Description automatically generated with low confidence

**8.2.2 Adding Git submodules to Github**

At this point all the changes you have done to create and add the Git submodule have been done locally only. The Git Submodule is only added to your local repository, but not yet to your remote repository.

We now need to commit and push all the changes we have done locally to our remote GitHub repository.

*git add .*

*git commit -m “Added sub”*

*git push -u origin*

After you successfully pushed your updated Lab 1 Git repository to GitHub, you should also see that the new Git submodules “Lab1\_submodule” folder has been added into GitHub

A screenshot of a computer program

Description automatically generated with low confidence  
**8.2.3 Working in a GitHub repository with multiple collaborators**

We have learned how to use Git and GitHub for a single user to locally and remotely respectively to archive our source code files and perform basic configuration management tasks to control the file versions, repository branches, etc.

However, most software projects are organized as teams of more than 1 developer, therefore Git and GitHub are usually used to manage the software codebase for a team of software developers.

**8.2.4 Adding collaborators in GitHub repositories**

Before we can start working with other developers, we need to grant the developers read and write access to allow them to commit and push files to our GitHub repository.

In your Lab 1 GitHub repository, go to “Settings” and select “Collaborators and Users”A screenshot of a computer

Description automatically generated with medium confidence

Click “Add people” and then add 1 of your classmates into your GitHub Lab 1 repository with the “write” role. Notice that there are several other roles that GitHub allows for collaborators.

A screenshot of a computer

Description automatically generated with medium confidence

**8.2.5 Committing and Pushing changes to shared GitHub repositories**

After your classmate has added you as a collaborator with “write” role access into their Lab 1 GitHub repository, you can already modify and add files to their GitHub repository.

The first step is to first clone your classmate’s Lab 1 GitHub repository locally onto your laptop. Change to C:\ drive and create a new empty folder “clone\_repo” (i.e. the directory is “c:\clone\_repo”). Open a CMD prompt, and change directory to this new folder using the command:

*cd c:\clone\_repo*

Next, clone the classmate’s Lab 1 GitHub repository with the Git command below.

*git clone <URL of repository to clone>*

After cloning your classmate’s Lab 1 repository, modify the Python file “HelloWorld.py” locally to add a second line to the file:

A screenshot of a computer

Description automatically generated with medium confidence

Commit and Push your modifications the “master” branch in your classmate’s GitHub repository for Lab 1.

*git push -u origin*

Finally you use the Git command below to view and verify the commit history in your classmate’s Lab 1 GitHub repository.

*git log*

**Question 1 : Why is it necessary to have “git add” in the git workflow?**

The git add command adds new or changed files in your working directory to the Git staging area. git add is an important command - without it, no git commit would ever do anything.

**Question 2 : What is the purpose of describing the changes in the commit command?**

So that you know what changes were made during each commit versions.

**Question 3 : What does the green and red colour coded line means when you issue git diff?**

Red is the old, Green is the new changes

**Question 4 : Under what circumstances will you use branching in a software development project?**

Git branches can be used for several different reasons, for dedicated bug fixes, new features, software releases, etc. Refer to notes

**Question 5 : What does the term “origin” refer to?**

In Git, "origin" is a shorthand name for the remote repository that a project was originally cloned from. The origin is the remote branch which is the primary working directory of a project.

**Question 6 : Do you need the “--set-upstream” option for subsequent git push commands?**

No. just need it once to set master branch

**Lab 2**

**1 Installation of Python 3 Interpreter**

Before we start to install the PyCharm Professional IDE, we first need to install the Python 3 interpreter which PyCharm will use for running our Python code.

**1.1 Download the Python 3 interpreter for Windows from the link below.**

Choose version 3.8.x (e.g., 3.8.10 for 64-bit systems)

<https://www.python.org/downloads/>

**1.2 Run the installer to start the Python 3 interpreter installation process.**

During installation, remember to tick “Add Python 3.8 to PATH” and “Disable path length limit”.

**2 Installation of PyCharm Professional Edition using “Educational License”**

**2.1 Download PyCharm version 2021.3.3 or higher from the following link below.**

Use your SP “iChat” email address to register. Choose PyCharm “Professional Edition”. You will use “educational license” at no cost for 1 year.

<https://www.jetbrains.com/shop/eform/students>

**2.2 Install the PyCharm “Professional Edition” using the educational license using your SP “iChat” email address.**

During the installation, tick the 4 options:

• Create Desktop shortcut

• Update PATH variable

• Update Context Menu

• Create associations

**3 Create a new PyCharm project**

**3.1 In your laptop’s D-drive, create a new folder “D:\ET0735\_Lab2”. You will use it to store your Python codes.**

**3.2 Launch the PyCharm software by double-clicking the shortcut on Desktop.**

A welcome window will pop up. Choose “New Project”.

**3.3 The “Create Project” dialog box will appear.**

Make sure that “Pure Python” has been selected. (Note: the color of your popup window may be different from what is shown in Figure 1 due to the different theme chosen for your PyCharm software.)

**3.4 For the “Location” field, click the folder icon at the far right.**

Choose the “D:\ET0735\_Lab2” folder you had created in Step 3.1, then click “OK”. This will set the location for storing your Python project.

A screenshot of a computer

Description automatically generated with medium confidence

**3.5 Click the small triangle icon ► next to “Python Interpreter: Python 3.8” (see Figure 1) to expand it (Figure 2).**

**3.6 Select the radio button to use “Previously configured interpreter”.**

For the “Interpreter” field, select only “Python version 3.8” (even if you have other versions on your laptop). Then, click the “Create” button at the bottom

A screenshot of a computer

Description automatically generated

**3.7 The PyCharm IDE will be launched, showing the newly created project Lab2.**

**3.8 While in the PyCharm IDE, you may click “Help→About” to see your PyCharm software licensing information.**

Check that you have correctly installed PyCharm “Professional Edition” and the expiry date is at least 1 year from the current date.

**4 Create and Execute Python scripts in a PyCharm project**

**4.1 In PyCharm IDE, right-click the “ET0735\_Lab2” project, choose “New”, and then select “Python File”.**

A computer screen shot of a computer

Description automatically generated with low confidence

**4.2 A popup window will prompt for the file name. Name the file “Lab2.py” and press the ENTER key.**

**4.3 Add the following Python code in the newly created Lab2.py**

A picture containing text, font, line, screenshot

Description automatically generated

**4.4 From the “Run” menu, select “Run…” to run Lab2.py code.**

The first time the “Run” option is selected in PyCharm, a dialog box will appear for you to select which Python script should be run and you should select “Lab2.py”.

Subsequently the “Run” option will execute the last run Python script without prompting you to select which script to execute.

**4.5 Check the console output in PyCharm.**

If your project was created correctly you should see the console output based on the initial code in Lab2.py.

A screenshot of a computer program

Description automatically generated with low confidence

**5 Create a Local Git Repository and Push to GitHub**

**5.1 Now, you will create a new local git repository to track your PyCharm Python project.**

**5.2 Open a Command Prompt window and change to the directory to “D:\ET0735\_Lab2”.**

**5.3 Enter Git command git init and check that the .git folder has been created in the “D:\ET0735\_Lab2”**

**5.4 Stage all the contents in the folder “D:\ET0735\_Lab2” using the git add \* command.**

The ‘\*’ represents a wildcard filter that requests git to stage all files within the current repository folder.

**5.5 After the staging is completed, commit to the local git repository.**

**5.6 Login to GitHub using the account you had created in Lab 1. Create a new repository at GitHub and name it Lab2. Copy the URL of the GitHub Lab2 repository. Refer to Lab 1 for the steps if necessary.**

**5.7 At your local git repository, specify the URL of the remote GitHub repository. Set up the upstream branch. After that, push the local git repository to GitHub.**

*git remote add origin https://github.com/kahnyee/LabT2.git*

*git remote -v*

*git push --set-upstream origin master*

*git push -u origin*

**5.8 Notice that not all the files located in the folder “D:\ET0735\_Lab2” are pushed to GitHub.**

To understand which files are ignored and not tracked in the local and remote repository, open the file .gitignore in “D:\ET0735\_Lab2\.idea” and study its contents to understand which files will be “ignored” by Git. These files ignored are created by the environment every time the file is opened therefore it would not be of use to track these files.

**6 Python Functions and Mathematical Operators**

**Python User Defined Functions**

To develop modular software code that is easy to maintain and extend, we need to decompose our Python code into functions.

Splitting the code into functions are also useful to simplify the coding implementation as well as to simplify the Software Unit Testing which will be covered in Lab 3.

Python is an indentation-based (each indentation is 4 spaces) programming language and functions are defined with the following syntax:

A picture containing text, screenshot, font, line

Description automatically generated

Notice in the above code we define a function starting with the keyword “def” followed by the function name and then a list of comma separated parameters. The code inside the function is indicated to start with a tab space (= 4 spaces), and the function can return some data using the “return” keyword.

**Python Arithmetic Operators**

The table below lists some of the most used Python arithmetic operators.

A picture containing text, font, screenshot, number

Description automatically generated

**Python Conditional Operators**

Please refer to the link below for examples of Python conditional operators.

<https://www.w3schools.com/python/python_conditions.asp>

**7 Python Main Entry Point, Console Input and String Processing**

**Python Main Entry Point**

In the previous exercise, we have implemented Python code that calculates the BMI value using a user defined function “calculate\_bmi”.

Also notice that we called the function “calculate\_bmi()” directly instead of a central “main” function or entry point.

While this works for a simple single Python file application, calling Python functions directly from a file might become messy if the application comprises of several Python files.

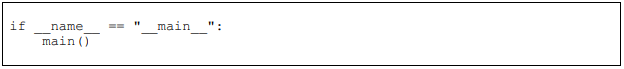
In this case it would be better to define a single Python file and “main” function as an entry point to start the entire application.

We should define the main entry point function “main()” which can then be used to call other user defined functions.

A picture containing text, font, line, screenshot

Description automatically generated

In Python, the code below checks if the current script is the main Python file and then calls the function “main()” that we have defined in the step above.



**Lab 3**

**1 Installation of PyTest in PyCharm**

* 1. **Launch the PyCharm software.**

If prompted to select a project, select the Lab2 project that you had created in Lab2 experiment.

* 1. **Go to “File → Settings”.**

This opens the Python Package window in PyCharm. Expand “Project: ET0735\_Lab2” and select “Python Interpreter”.

A screenshot of a computer

Description automatically generated with medium confidence

**1.3 Under “Python Interpreter”, check if the PyTest Python package has already been installed.**

For the example shown in Figure 1, there is no sign of PyTest Python package. This means that PyTest has not been installed.

**1.4 If PyTest has not been installed, then click the “+” icon to add new package.**

A screenshot of a computer

Description automatically generated with medium confidence

**1.5 The “Available Packages” window pops up, listing packages available to be installed.**

In the search textbox, enter “pytest”. The PyTest package will be shown and selected. Click the “Install Package” button to install PyTest.

A screenshot of a computer

Description automatically generated with medium confidence

**1.6 After the installation of the PyTest package has completed, check that PyTest is**

**now listed under the Python Interpreter installed packages list**

(i.e., repeat Step 1.2). You should find Pytest in the list.

Steps 1.5 and 1.6 can be used to install most of the 3rd party additional Python libraries you might need in the future.

**2 Configure PyTest for Unit Testing in PyCharm**

After the PyTest package has been installed, we need to set up the current PyCharm project to specify that we will use PyTest for all the Unit Tests defined within the current PyCharm project scope.

**2.1. Go to “File → Settings → Tools → Python Integrated Tools”.**

In the “Testing” section, open the dropdown list of the “Default test runner” field, and select “pytest”. Click “Apply”, then click “OK”. This now configures PyCharm to use PyTest as the default Unit Testing Tool.

A screenshot of a computer

Description automatically generated with medium confidence

**3 Create and Execute Software Unit Tests using PyTest**

In the previous lab we learned the basics of Python programming, we will now extend this to also introduce the basics of Software Unit Testing based on the Python PyTest Unit Testing framework.

The PyTest Unit Test framework uses standard Python code to define Unit Test cases which we can use to test each Python function we have implemented.

Using PyTest, we use the keyword “assert( … )” to evaluate the return value of a Python function with the expected value/s returned by the function under test.

Follow the steps below to clone an example of a sample Python script that sorts some numbers in ascending and descending order,

**3.1. Go to the D:\ directory of your laptop. Create a new folder “ET0735”. If you already have one, skip this step.**

**3.2. Open a CMD prompt window, and change directory to D:\ET0735.**

**3.3. Clone the Lab 3 Git repository from the link https://github.com/ET0735-DevOpsAIoT/Lab3.git, using the git command below.**

*git clone https://github.com/ET0735-DevOps-AIoT/Lab3.git*

**3.4. A new folder “Lab3” will be created in the D:\ET0735 directory. This folder is a clone of the remote GitHub repository that you had specified in Step 3.3. You should find the following 5 items in the Lab3 folder:**

• .git

• .idea

• README.md

• Lab3.py

• Test\_Lab3.py

**3.5. To open the newly cloned project for experiments, in PyCharm, click “File → Open”. When the “Open File or Project” window pops up, navigate to D:\ET0735, select the Lab3 folder and click “OK”.**

**3.6. When prompted about how to display this project, choose “This window”. The project Lab2 will be closed, and project Lab3 will be opened.**

**3.7. Right-click “Lab3.py” and select “Run Lab3” from the dropdown list. This runs the Python file Lab3.py. Check that the correct console output is displayed to sort a list of numbers in ascending or descending order, as shown below.**

A screen shot of a computer program

Description automatically generated with low confidence

**3.8. In the PyCharm project Lab3, notice that there is an additional file “Test\_Lab3.py”. Double-click it to open.**

A screenshot of a computer

Description automatically generated with medium confidence

“Test\_Lab3.py” is a Python file where we have defined the PyTest Unit Test to check that all the functions implemented in “Lab3.py” are correct

In “Test\_Lab3.py”, the following PyTest Unit Test cases are defined,

- Test Case 1 → test\_bubble\_sort\_ascending()

- Test Case 2 → test\_bubble\_sort\_descending()

- Test\_Case3 → test\_bubble\_sort\_invalid()

Test Cases 1 and 2 are known as “Positive” test cases where the test cases check for valid input combinations versus the expected result.

Test Case 3 is for checking what happens when invalid or unexpected inputs are passed into the function “bubble\_sort()”.

**3.9. For each PyTest test, notice the format and syntax below where each test cases ends with an “assert” statement**

A screenshot of a computer code

Description automatically generated with low confidence

The “assert” statement in PyTest basically returns a Boolean value True or False of the condition asserted or checked is True or False.

In Software Unit Testing, we basically want to verify and check that the function under test returns some predefined expected values based on a set of corresponding inputs.

**3.10. In the PyCharm project Lab3, notice that there is an additional file “Test\_Lab3.py”. Double-click it to open.**

To execute PyTest unit test cases inside the PyCharm IDE, there are 2 main methods,

- Execute all PyTest Unit Test cases within a Python script

- Execute selected individual test cases

If you want to execute all PyTest test cases, just right click on any PyTest Python file (see Figure 11) to run all test functions within the selected Python file.

A screenshot of a computer

Description automatically generated

To run selected PyTest functions individually in PyCharm, just click on the green icon on the left margin of the PyTest function you want to execute (see Figure 12).

A screenshot of a computer

Description automatically generated with medium confidence

**3.11. Try to execute all PyTest test cases in the Test\_Lab3.py file, using the approach shown in Figure 11.**

After running all the PyTest Unit Test case, a Test Status report will be shown in PyCharm, summarizing the test cases that were executed and the test results.

A screenshot of a computer

Description automatically generated with medium confidence

**In order for git to ignore certain specific files or folders, you will need to add a new file .gitignore.**

In your code folder create a new file .gitignore. To ignore \_\_pycache\_\_ folder, you can add the following code in .gitignore txt file:

\_\_pycache\_\_/

**Update the remote origin to the URL of your ne Github repository using the command below replacing {URL} with the URL to your new Github “Lab3” repository**

*git remote set-url origin {URL}*