# Software for Digital Innovation (CIS4044-N) Tutorial 5: Consolidation and Git

## Before You Start

There are many excellent Git hosting providers out there. Two popular ones used by staff within the University are Bitbucket and GitHub. Research and pick one to sign up to.

**Hint:** For this module it is recommended for you to create a GitHub account. As a student you have access to the GitHub Student Developer Pack available at: <https://education.github.com/pack>

The pack provides free access to some of the “best developer tools” in one place so you can learn by doing, as well as unlimited private repositories.

Note you will need to verify you student status. First, create a GitHub account with your *personal* email address, after which you can link your university email address to gain access to the Student Developer Pack.

## Introduction

This session aims to consolidate your learning so far whilst introducing you to Git version control.

## Activity 1: Consolidation

Go back through the previous four tutorials and ensure you have completed *every* question.

If you have any queries please seek help from your peers or tutor, do not sit in silence. You will need Python to click at some point in order to complete the ICA, this will only happen if you engage with the module and seek help when needed.

## Activity 2: Useful Python Scripts

By now you have built up a large amount of python scripts that will help you build towards an effective solution to ICA parts 2 & 3.

Let’s remove the annoyance of copying to and from nethome, and instead collate our Python programs into a single directory and use Git with a remote server to manage them.

Using Week 5’s lecture slides as a guide do the following:

1. Create a new directory called useful\_python.
2. Move into the useful\_python directory and initialise a new local git repository.
3. Create a new file hello.py that prints “Hello World!”
4. Stage hello.py and create an “Initial commit”.
5. Using your remote Git provider of choice, create a new repository called “useful-python”. Note, do not initialise a remote repository with a README if you already have a local you are intending to sync.
6. Link the remote repository to your local one.
7. Do a git push to sync your local repository with the remote.
8. Confirm this is successful on the remote providers web interface.
9. Go through your current Python programs. You should have a large number from the tutorials, and do not forget the code samples uploaded each week. Select an example that best represents a key feature of the Python language (i.e. operators, lists, functions, etc.) and copy it to your useful\_python directory. Note you may want to rename them to match the feature.
10. Stage the new file and commit (with an appropriate commit message).
11. Repeat steps 9 and 10 until you are satisfied with your collection of Python examples.
12. Do a git push to sync your local repository with the remote.
13. Confirm this is successful on the remote providers web interface.
14. Delete the local folder useful\_python.
15. Clone the remote repository created earlier and confirm that no work has been lost.

## Extension Activity: Sorting

Pseudocode of several popular educational sorting algorithms is provided. Attempt to implement each in Python by taking a list of integers and sorting them smallest to largest.

**Note:** This activity is designed to be challenging! Only attempt this if you have done all previous activities. You are free to research the algorithms in more detail, but please refrain from simply googling Python implementations.

1. **Bubble sort** is a simple sorting algorithm that repeatedly steps through a list, compares adjacent elements and swaps them if they are in the wrong order. The pass through the list is repeated until the list is sorted.

The algorithm, which is a comparison sort, is named for the way smaller or larger elements "bubble" to the top of the list.

Suppose A is a list of N values. We want to sort A in ascending order.

For I = 0 to N - 2

For J = 0 to N - 2

If (A(J) > A(J + 1)

Temp = A(J)

A(J) = A(J + 1)

A(J + 1) = Temp

End-If

End-For

End-For

1. **Insertion sort** is a simple sorting algorithm that builds the final sorted list one item at a time.

Insertion sort iterates, consuming one input element each repetition, and growing a sorted output list. At each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there. It repeats until no input elements remain.

Suppose A is a list of N values. We want to sort A in ascending order.

For I = 1 to N-1

J = I

Do while (J > 0) and (A(J) < A(J - 1)

Temp = A(J)

A(J) = A(J - 1)

A(J - 1) = Temp

J = J - 1

End-Do

End-For

1. **Selection sort** is an in-place comparison sorting algorithm.

The algorithm divides the input list into two parts: a sorted sub-list of items which is built up from left to right at the front (left) of the list, and a sub-list of the remaining unsorted items that occupy the rest of the list.

Initially, the sorted sub-list is empty, and the unsorted sub-list is the entire input list. The algorithm proceeds by finding the smallest element in the unsorted sub-list, exchanging (swapping) it with the leftmost unsorted element (putting it in sorted order), and moving the sub-list boundaries one element to the right.

Suppose A is a list of N values. We want to sort A in ascending order. That is, A[0] should be the smallest and A[N-1] should be the largest.

For I = 0 to N-1 do:

Smallsub = I

For J = I + 1 to N-1 do:

If A(J) < A(Smallsub)

Smallsub = J

End-If

End-For

Temp = A(I)

A(I) = A(Smallsub)

A(Smallsub) = Temp

End-For

1. **Bogosort** (also known as permutation sort, stupid sort, shotgun sort, or monkey sort) is a highly *inefficient* sorting algorithm based on the generate and test paradigm.

The function successively generates permutations of its input until it finds one that is sorted. Note, please never use this in a real system!

Suppose A is a list of N values. We want to sort A in ascending order.

While not isInOrder(A):

shuffle(A)

1. Devise a way of generating random integer lists of varying length to test the efficiency of your algorithms.

## Document History

Revision 0 (25-Oct-20): This is the initial version of the 2020/21 exercise.