COS1000
Computational Thinking for Scientists

Introduction

- Computational thinking

  It is the thought process involved in formulating a problem and expressing a possible solution in such a way that a human or a machine can effectively carried it out.

- Coding using Python

  You will learn how to code using Python programming language because that's the most concrete way of appreciating the computational thinking steps.

- Basic mathematics.

  There is no computation without mathematics. Computation = taking some input and delivers some output, following some unambiguously prescribed steps of calculation.

Module Description

This module introduces computational thinking as applied to problems in science.

A selection of examples will be chosen to illustrate

- problem formulation and solution development:
  - abstraction – identifying essentials and weeding out less relevant details
  - decomposition – breaking the problem into smaller, more manageable sub-problems
  - pattern recognition — where have we seen similar problem(s) before? Can we apply the same solution technique?
  - Algorithms – develop a step-by-step process to solve the problem so that the work is replicable by humans or computers.
- analysis of the computational solutions and data visualization.

Module Description cont'd

- The selection will tackle different types of approaches typically used in scientific computational thinking, including deterministic and probabilistic methods
- Computers and software:
  - We will be using the Python programming language and the Jupyter Notebook interface.
  - You may need to install the open source Python software onto your laptop so that you can work on your projects anywhere anytime.

Classes of Illustrative Problems

Deterministic Problems

Typically, such problems are formulated in the form of difference or differential equations. Solving such equations gives deterministic solutions

- Population dynamics in ecological studies can be formulated using difference equations, with forecasts predicted by simple iterations. Both periodic and chaotic long-term behaviours can emerge

- Model of the spreading of diseases will be formulated as coupled differential equations.

Classes of Illustrative Problems cont'd

Probabilistic Solutions

Many scientific problems involve elements of probability, the prime example of which is the concept of randomness. The solutions (typically obtained using computer simulations) to such problems provide the averages as answers. Simple Monte Carlo methods (e.g. estimating the area of a circle) provide an exposure to statistical approaches, and the notions of sampling, averages, and convergence.

Another example is the traffic simulation, which can be tackled using discrete event simulation (DES) methods.

Grading

- Grading for the module will be based on continuous assessment components:

Continuous assessments	Weightings
Lab Notebooks	20%
Online quizzes	5%
Two Term Tests	40%
Two Projects	35% (15% and 20%)

Online quizzes on LumiNUS: MCQ

Two term tests during lecture slots. Students are to work on them individually, without discussing with one another.

Project 1 (15%) will cover content from lectures and labs.

Project 2 (20%) will contain tasks with a significant design component, and graded partly according to how well the code performs.

Projects 1 and 2: Students will work on them in groups of three

Lecture & Tests schedule (tentative)

Week	Topics to be covered
1-	Introduction to COS1OOO and module logis-tics
	Population dynamics – the Logistic map
	Lotka-Volterra Predator-Prey model
	Spreading of diseases
5	Project 1

Week	Topics to be covered
7	Test1 (Fri)
7-	Random numbers and Monte-Carlo estimation
9	Project 2
12	Term Test 2 (Fri)

Python

On Python programming:

- J M Stewart, Python for Scientists, 2nd Edition (2017) [NUS eBook]
- J M Kinder & P Nelson, A Student's Guide to Python for Physical Modeling (2018)

Jupyter notebooks

Introduction to Jupyter notebook

- It is a web-based interactive computing platform that allows users to author documents that combine live code, equations, narrative text, interactive dashboard and other rich media, with data analysis executed in real time.
- These Jupyter notebook documents (.ipynb extension) are basically the electronic version of science practical books which can be easily communicated with others using email, etc.
- Jupyter Notebook is being used in all areas of academic research (UC Berkeley, Stanford, etc.) as well as the industry (Google, IBM, Facebook, Microsoft, etc.).

Notebook Kernel

- A notebook kernel is a “computational engine” that executes the code contained in a notebook document.
- When you open a Notebook document, the associated kernel is automatically launched. When the notebook is executed, the kernel performs the computation and produces the results
- Depending on the type of computations, the kernel may consume significant CPU and RAM. Note that the RAM is not released until the kernel is shut down.
- Tip: If your Jupyter notebook just refuses to run properly, restarting the Kernel may be what is needed.

Getting Started: Installation of Jupyter Notebook

To get started, the first step is to install Jupyter Notebook. To many first-time programmers, this itself can be a hurdle because there are various possible ways to go about doing that, some not that straightforward.

For the most transparent installation experience, we recommend installing Jupyter via Anaconda Navigator, which encompasses an open-source suite of programming languages for scientific computing (e.g. Python), with streamlined programming logistics i.e. package management and deployment. A detailed installation guide can be found here, with a helpful step-by-step video guide.

This installation process should not take more than a few minutes, although it may be much slower on systems with certain processes i.e. virus scans running in the background.

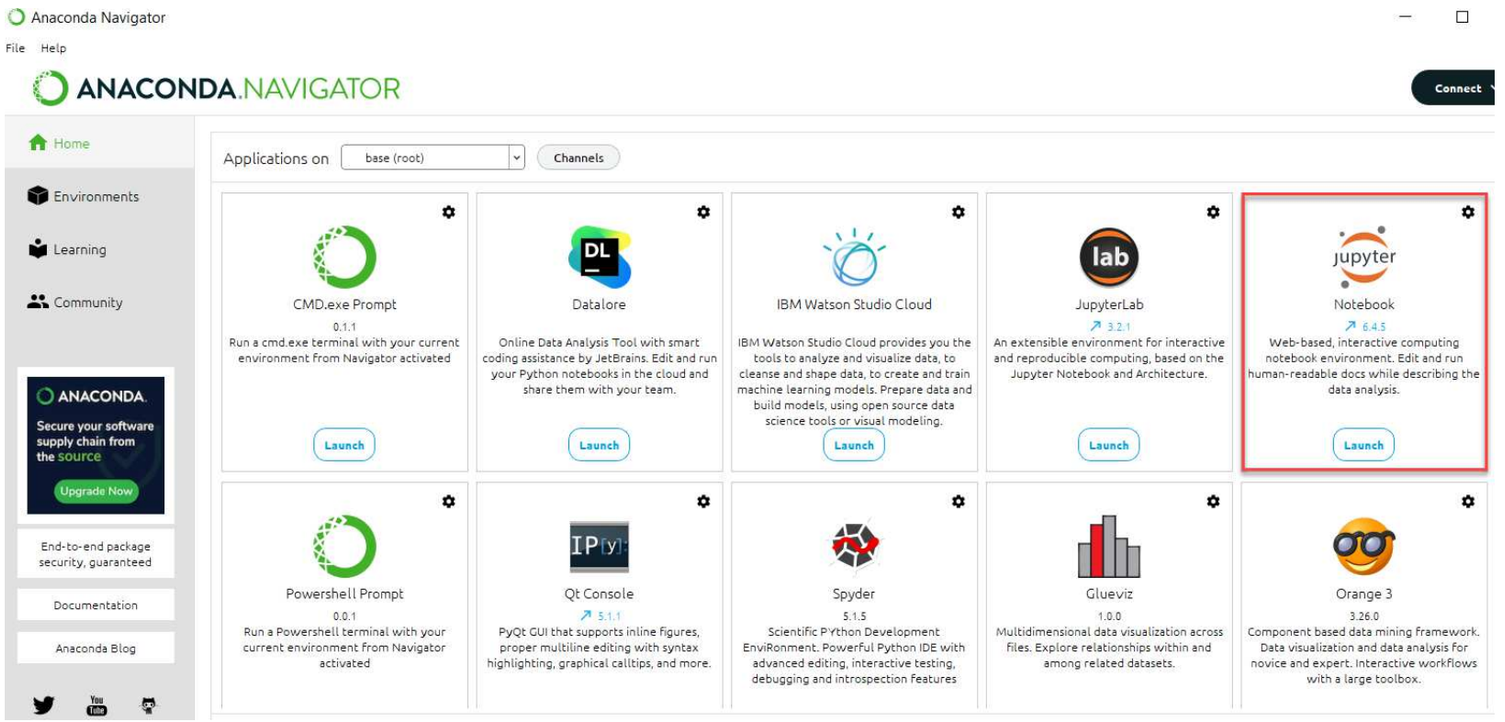
Launching the Jupyter Notebook

Now that the installation is done, the next step is to get your Jupyter Notebook fired up and running

- To launch Jupyter notebook:
  - Windows version: open Anaconda Navigator which should be in your start menu. Alternatively, you can instead launch Anaconda Prompt (Anaconda3), and start a notebook server from the command line by typing “jupyter notebook”
  - MacOS version: You are strongly encouraged to use Anaconda Navigator to launch your jupyter notebook instead of console (terminal). Open Launchpad, and then click the Anaconda Navigator icon.

Launching the Jupyter Notebook cont'd

- After launching Anaconda Navigator, you should see the following desktop graphical user interface (GUI) as shown in the figure below:



Next, click on the Launch button on the Jupyter Notebook tile (see figure above).

Launching the Jupyter Notebook cont'd

- This should automatically launch the following page on your default web browser. Its URL, which should by default look something like (localhost:8891/tree), is not a weblink, but rather refers to your local root (home) directory corresponding to the folder from which the network server was first started. This is the location of your Jupyter workspace, which is on your computer, not a web server or cloud location!

Notebook dashboard - Homepage of the Jupyter notebook web application. This Notebook dashboard is the "file browser" for your Jupyter workspace.

Computational Thinking

Computation and Simulation

With the availability of fast computers, efficient algorithms and visualization tools, scientific investigations can actually be performed away from the lab.

- Some experiments can never be performed, for example:
  - evolution of the universe
  - epidemic spreading
- Some experiments can be very expensive to conduct, for example:
  - new aircraft design
  - drug design

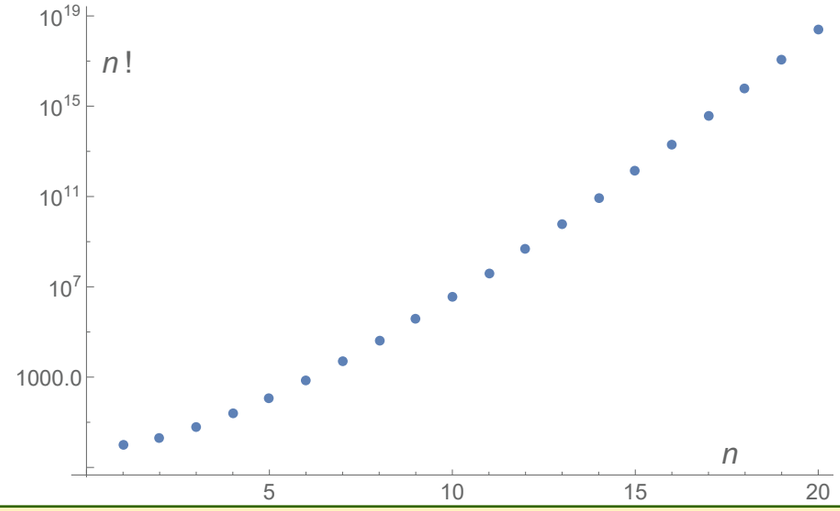
Computations – Coding & Computational Thinking

Computations on a computer requires us to give instructions to computing machines.

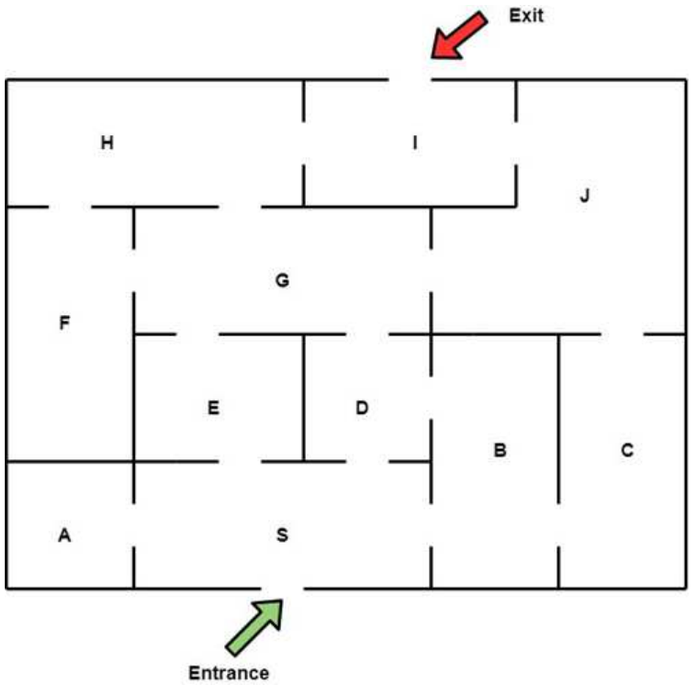
- Coding is that precise/unambiguous communication, hence the most concrete way to realize the computational thinking steps.
- Coding needs algorithms (step-by-step recipes) to implement the computation.
  - For example, how does one compute the factorial of a positive integer $N! = 1 \times 2 \times \cdots \times (N-1) \times N$?

Computations – Coding & Computational Thinking cont'd

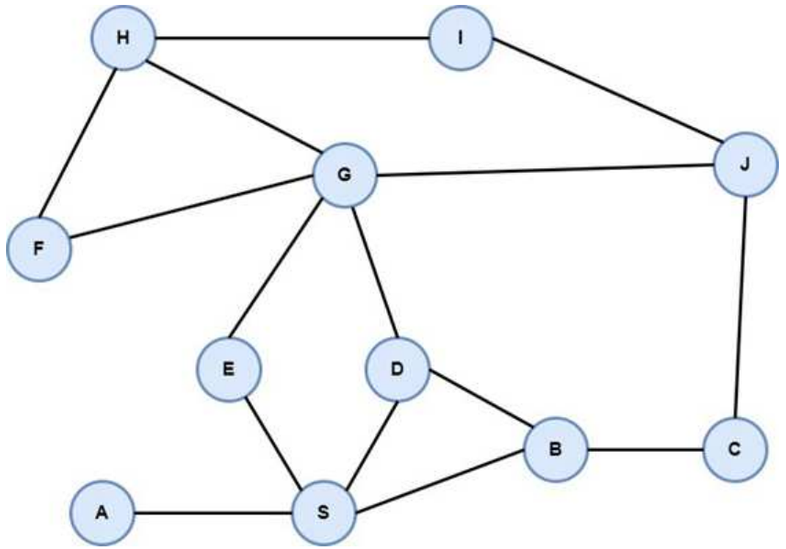
- It is necessary to verify that the computer actually performs the computations we want it to, and that it does so correctly.
  - Make use of past experience, or verify with known special cases, e.g. we know $1! = 1$ and $2! = 2$, $3! = 6$.
- Graphical visualization helps with assessing the "reasonableness" of the computations.



An example: Maze



Given the maze above, we want to navigate from entrance to exit.

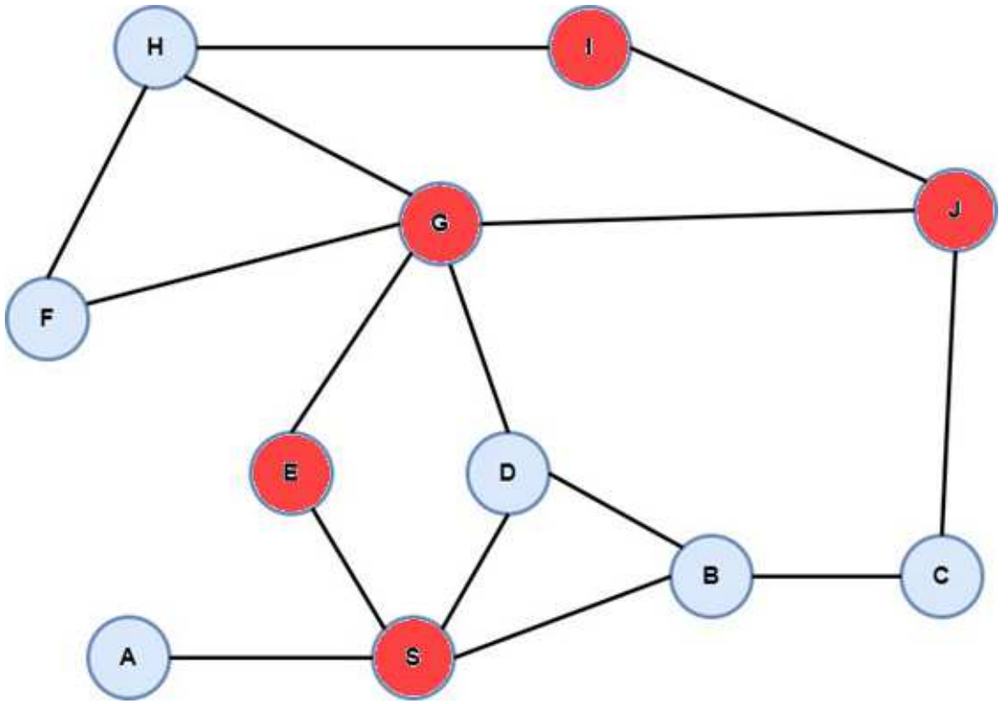


We want to model it into a graph. We can create a vertex for each room and an edge for each door of the maze. In all, we have 11 vertices (rooms) and 15 edges (doors)

An example: Maze cont'd

Depth first search

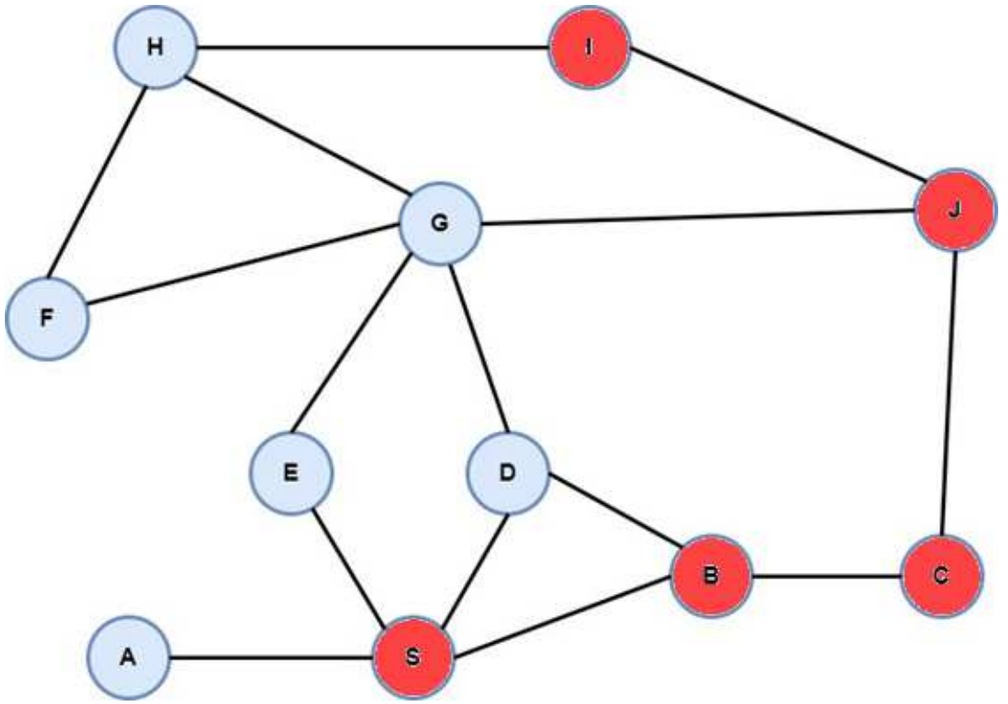
1. Start from the entrance
2. Arbitrarily pick one of its neighbours and go there
3. If that node has neighbours, pick any one of those and go there unless it has already been visited
4. Repeat steps 2–3 until one of two things happens: (i) If target node is reached, we terminate the algorithm and report success, or (ii) if a node is reached with its neighbours already been visited, or no neighbours at all, we backtrack and try one of the neighbours we haven't visited before



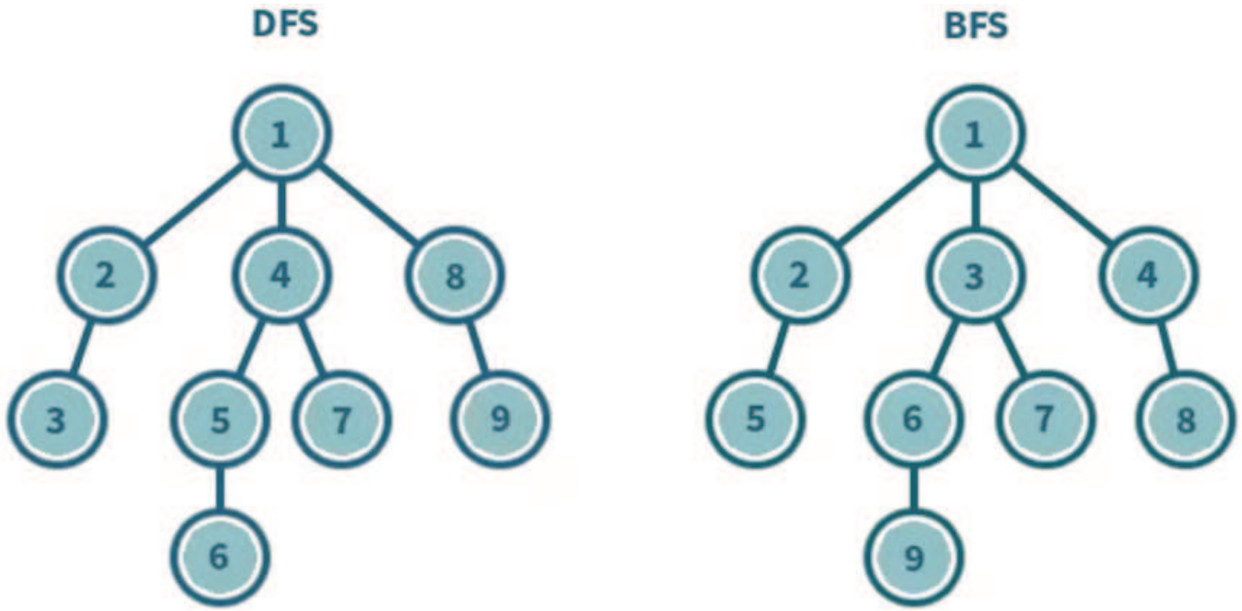
An example: Maze cont'd

Breadth first search

1. Start from the entrance
2. Visit all neighbours of this node
3. Then visits the neighbours' neighbours and so on



An example: Maze cont'd



DFS (Depth first search): The search goes deep into a particular path, and tries other options only if a dead end is met

BFS (Breadth first search): The search explores several paths simultaneously till the endpoint is found