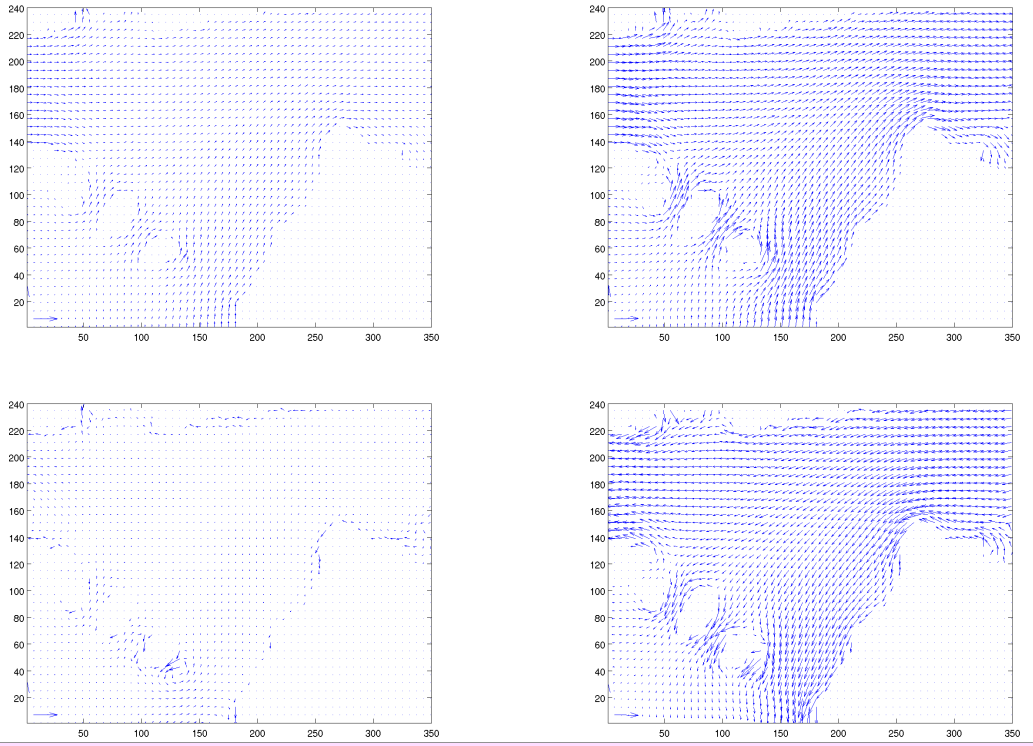
CS319: Scientific Computing

Introduction to CS319

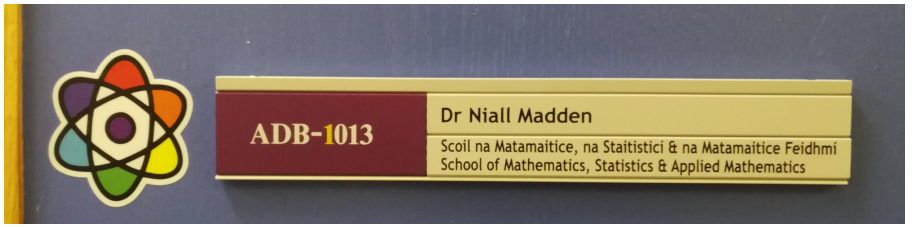
Dr Niall Madden

Week 1: 14+15 January, 2026



0. Outline

Lecturer details:



Who: Dr Niall Madden (he/him)
How to greet: Niall ("Knee-al" `#StartsWithAName`)
From: School Mathematical and Statistical Sciences.
Where: Office: AdB-1013, Arás de Brún.
Contact: Email: Niall.Madden@UniversityOfGalway.ie

Students: 3rd year Mathematics and/or Computing/Data Science (3BS9); 3rd and 4th year Mathematical Science (3BMS2+4BMS2), 4th Year Maths/Applied Maths (4BS2); 1(?) Erasmus; 2(?) Study Abroad; 1 PhD student.

This module involves a mix of lecture "classes" and lab sessions.

- The "classes" will mix conventional lectures, and practical sessions. That is, you will spend some time coding in every class.
- All slides and lecture materials (such as sample programmes) will be made available in before the Wednesday class.

	Mon	Tue	Wed	Thu	Fri
9 – 10					
10 – 11					
11 – 12					Lecture
12 – 1					
1 – 2					
2 – 3					
3 – 4					
4 – 5			Lecture		

Tentative plan: Labs Thursday at 9 or Friday at 12.

We'll use Canvas for

- Posting announcements (1 per week, usually);
- Posting grades
- Assignment uploads
- *Links* to slides and scripts from class.
  All materials will actually be hosted at
  `https://www.niallmadden.ie/2526-CS319`

Code (and there will be lots of it) can also be made available on a `git` repository. If interested, talk to Niall.

The notes for CS319 are largely self-contained. But some books will be VERY helpful. The reading list is at https://nuigalway.rl.talis.com/modules/cs319.html but will be updated. Key books include

- Scientific Computing with Case Studies (Diane O'Leary) https://epubs-siam-org.nuigalway.idm.oclc.org/doi/book/10.1137/9780898717723
- Practical C++ programming (Steve Oualline). https://search.library.nuigalway.ie/permalink/f/1pmb9lf/353GAL_ALMA_DS515666310000362 6
- Think Python (Allen B. Downey) https://greenteapress.com/wp/think-python-3rd-edition
- More will be added ...

The final grade for CS319 will be based on

- **Four programming assignments**, worth 10% each, with deadlines in Weeks 5, 7, 9 and 11.
- a mid-semester open-book test, worth 20%, in **Week 6** (Discuss!) → Probably Friday 11am.
- a project and presentation (40%); presentations on Wed 25th and one other time (TBC).

This module does not have an end-of-semester exam.

Check back later for notes on the policy for use of GenIA (spoiler: not allowed, unless answering a question **about** GenAI).

2. CS319: what and why

In CS319 we are primarily concerned with *three* issues:

1. How to use a computer to solve a scientific problem. That is:
   - how to determine the best algorithm to apply in a given situation.
   - how to understand the potential and limitations of the algorithm.
2. Implementing that algorithm: How to write the code! Not just that, we'll also learn how to code in C++. (Why C++? More on that later...)
3. Testing/verifying/validating the implementation (for correctness and efficiency).

2. CS319: what and why

More deeply, this is a course on **programming and problem-solving**.

It is NOT a "first course on programming". You are expected to be proficient in at least one language. For most of you, that language is Python. For some it is Java. (Any others?)

The primary learning outcomes are that, by the end of the semester, you can honestly list

- "Skilled in scientific computing"
- "Can programme in C++"

on your CV.

There are various good candidates for a language with which to do Scientific Computing, including

- Python
- MATLAB/Octave
- C/C++
- Julia
- others?

Python: advantages

(i) Lots of great libraries, specially `NumPy`, `scipy`, and `matplotlib`;

(ii) Free!

(iii) Good IDEs/ notebooks.

(iv) Many people already have expertise.

(v) ...

Python: disadvantages

(i) Can be very slow.

(ii) Some of you are already quite expert, some less so.

(iii) ...

MATLAB/Octave: advantages

(i) Specifically designed for SciComp
(ii) Fast!
(iii) Lots of tools/libraries included; good at visualisation.
(iv) Excellent IDEs and notebook environment
(v) Everyone starts from the same place.
(vi) ...

MATLAB/Octave: disadvantages

(i) MATLAB is expensive
(ii) Skills not so transferable.
(iii) ...

- (NN: check?)

C++: advantages

(i) Fast!
(ii) Valuable for your CV
(iii) Transferable skills (e.g, Arduino)
(iv) Everyone starts from the same place (right?).
(v) Free
(vi) ...

C++: disadvantages

(i) Very few libraries; no standard IDE
(ii) Steep learning curve
(iii) More time needed for studying the language
(iv) ...

So we'll make use of both C++ and Python:

- We'll learn the basics of coding in C++
- Our code will run very efficiently
- We'll import results into Python/Jupyter for further analysis.

You can use any Jupyter solution you like. To use the School server, watch out for an email from `maths-sto@universityofgalway.ie` then use the data provided to access `https://cloudjupyter.universityofgalway.ie/`

3. Scientific Computing

Dianne O'Leary describes a **computational scientist** as:
"*...someone whose focus is the intelligent development and use of software to analyse mathematical models*".

These models arise from problems formulated by scientists and engineering. Solutions/models can then be constructed using statistics and mathematics. Numerical methods are then employed to design algorithms for extracting useful information from the models.

3. Scientific Computing

In scientific computing, we are interested in the **correct**, **reliable** and **efficient** implementation of these algorithms. This requires knowledge of how computers work, and particularly how numbers are represented and stored.

History has shown that mistakes can be very, very costly.

Source: Wikipedia

For us, the major topics of CS319 will be

- **Computer representation of numbers**, as well as more complicated objects, such as vectors and matrices.
- Defining functions (of various types);
- Differentiation and integration of (mathematical) functions.
- **Root-finding and optimisation**
- Efficiency and complexity of algorithms (from an experimental/applied view).
- Solving linear systems by direct and iterative methods
- Etc.

4. A first example

In the first few weeks of the module, we'll emphasise C++ more that Scientific Computing.

First, though, we are going to study, without too much explanation, how to implement a simple algorithm in each of the three languages mentioned earlier: Python, MATLAB, and C++. We'll also estimate their (in)efficiency.

The problem is to write some code that will sum all the elements in a list, and report how long it took.

In each case, we'll take the simplest possible approach, and ignore that each of these languages has (somewhat built-in) functions to do this.

TimeAlg1.py

```
# Sum the elements of a list in Python
import time

N=10**8      # N=10^n
A = [1]*N
start = time.time()
s1=0;
for i in range(len(A)):
    s1+=A[i]
t1 = time.time() - start
print(f"N={N:6.0e}, error={s1-N}, time(s)={t1:6.2f}")
```

TimeAlg1.m

```
% Sum the elements of a link in MATLAB/Octave
N = 10^8;    % N=10^n
A = ones(1,N);
start=tic;
s1 = 0;
for i=1:length(A)
    s1=s1+A(i);
end
t1=toc(start);
fprintf('N=%8.2e, error=%d, time(s)=%8.4f\n',...
         N, s1-N, t1)
```

TimeAlg1.cpp

```
#include <iostream>
#include <time.h>
#include <math.h>
int main() {
  int N=pow(10,8); // N=10^n
  double *A = new double [N];
  for (int i=0; i<N; i++)
    A[i]=1.0;
  clock_t start=clock();
  double s1=0;
  for (int i=0; i<N; i++)
    s1+=A[i];
  double num_clocks = (double)(clock()-start);
  double t1 = num_clocks/CLOCKS_PER_SEC;
  std::cout << "N=10^" << log10(N)
            << ", error=" << s1-N
            << ", time(s)=" << t1 << std::endl;
  return(0);
}
```

5. Introduction to C++

Adapted from Wikipedia

C++ (pronounced "C plus plus") high-level, general-purpose programming language created by Bjarne Stroustrup. First released in 1985 as an extension of C programming language, but as an object-oriented language.

In the TIOBE Index for Jan 2026, C++ is ranked as the 4th most popular language, behind Python, C, and Java, and just ahead of C#, and JavaScript... It is well ahead R (10), MATLAB (14).

Finished here Wed @ 5pm

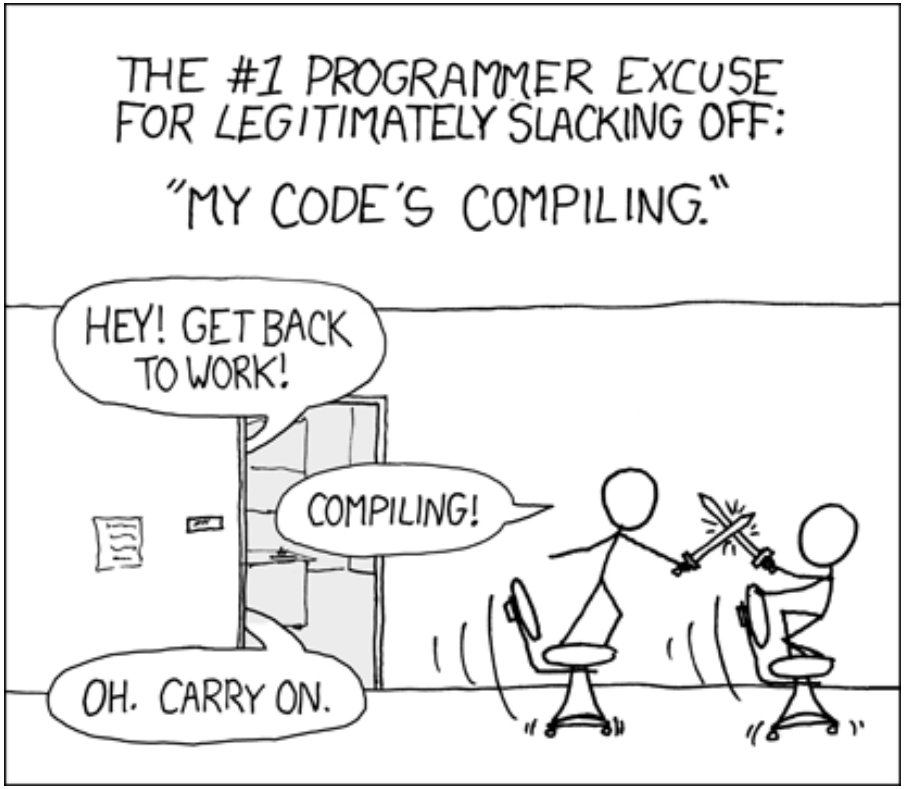
5. Introduction to C++

The main difference between C++ and (say) Python is the C++ is exclusively a **compiled** language (and not interpreted):

- Write the code
- Compile the code into an executable file.
- Run the executable.

C++ does not have a interactive REPL (*Read, Evaluate, Print, Loop*).

(Don't worry - this will be funny by March).



Source: https://xkcd.com/303

The C++ topics we'll cover are

1. From Python to C++: input and output, data types and variable declarations, arithmetic,
2. loops, Flow of control (`if` statements), conditionals,
3. functions.
4. Arrays, pointers, strings, and dynamic memory allocation.
5. File management and data streams.
6. Introduction to classes and objects.

To get started, we'll use an online C++ compiler. Try one of the following

- https://www.onlinegdb.com
- http://cpp.sh
- https://www.programiz.com/cpp-programming/online-compiler/

On AdBG021 we can also use Code::blocks.

On your own device, try installing one of the following free IDE's and compilers.

- Windows: `Code::blocks` (install `codeblocks-25.03mingw-setup.exe`)
- maxOs: Xcode
- Any: VScode
- Linux: it is probably already installed!

The convention is the give C++ programs the suffix *.cpp*, e.g., *hello.cpp*. Other valid extensions are *.C*, *.cc*, *.cxx*, and *.c++*.

If compiling on the command line with, e.g., the GNU Project's C/C++ compiler, the invocation is

```
$ g++ hello.cpp
```

If there is no error in the code, an executable file called `a.out` is created.

The workflow is different with an IDE: we'll demo that as needed.

Most/all of you have some familiarity with Python. There are numerous resources that introduce C++ to Python-proficient programmers.

For example: https://runestone.academy/ns/books/published/cpp4python/index.html One of its advantages is that it allows you to try some code in a browser.

Let me know if you find any other useful resource.

6. Basic program structure

- A "header file" is used to provide an interface to standard libraries. For example, the *iostream* header introduces I/O facilities. Every program that we will write will include the line:

```
#include <iostream>
```

Python Comparison

This is a *little* like `import` in Python.

- Like Python, the C++ language is case-sensitive. E.g., the functions `main()` and `Main()` are not the same.

6. Basic program structure

- The heart of the program is the `main()` function – every program needs one. When a compiled C++ program is run, the `main()` function is run first. If it is not there, nothing happens!
- "Curly brackets" are used to delimit a program block.

Python Comparison

This is similar to the use of "*colon and indentation*" in Python.

Example:

```
for i in range(10):
    x = x+i
```

```
for (i=0; i<10; i++)
{
   x= x+i;
}
```

6. Basic program structure

- Every (logical) line is terminated by a semicolon;
  Lines of code not terminated by a semicolon
  are assumed to be continued on the next line;

- Two forward-slashes // indicate a comment – everything after them is ignored until an end-of-line is reached.

Python Comparison

So, this is similar to a # in Python.

$x = y + z + 6j \quad q = 3j;$

```
x = y +
  z +6j
```

6. Basic program structure

- Every (logical) line is terminated by a semicolon;
  Lines of code not terminated by a semicolon
  are assumed to be continued on the next line;

- Two forward-slashes // indicate a comment – everything after them is ignored until an end-of-line is reached.

Python Comparison

So, this is similar to a # in Python.

```
x++;  // same as x=x+1 and x+=1
```

Also, can use /* STUFF */ for comments (like a doc-string).

This program will output a single line of output.

00hello.cpp

```
#include <iostream>
int main()
{
  std::cout << "Howya␣World.\n";
  return(0);
}
```

Finished here Fri at 12

00hello.cpp

```
#include <iostream>
int main()
{
  std::cout << "Howya␣World.\n";
  return(0);
}
```

- the identifier `cout` is the name of the **Standard Output Stream** – usually the terminal window. In the programme above, it is prefixed by `std::` because it belongs to the *standard namespace*...
- The operator `<<` is the **put to** operator and sends the text to the *Standard Output Stream*.
- As we will see `<<` can be used on several times on one lines. E.g. `std::cout << "Howya World." << std::endl;`

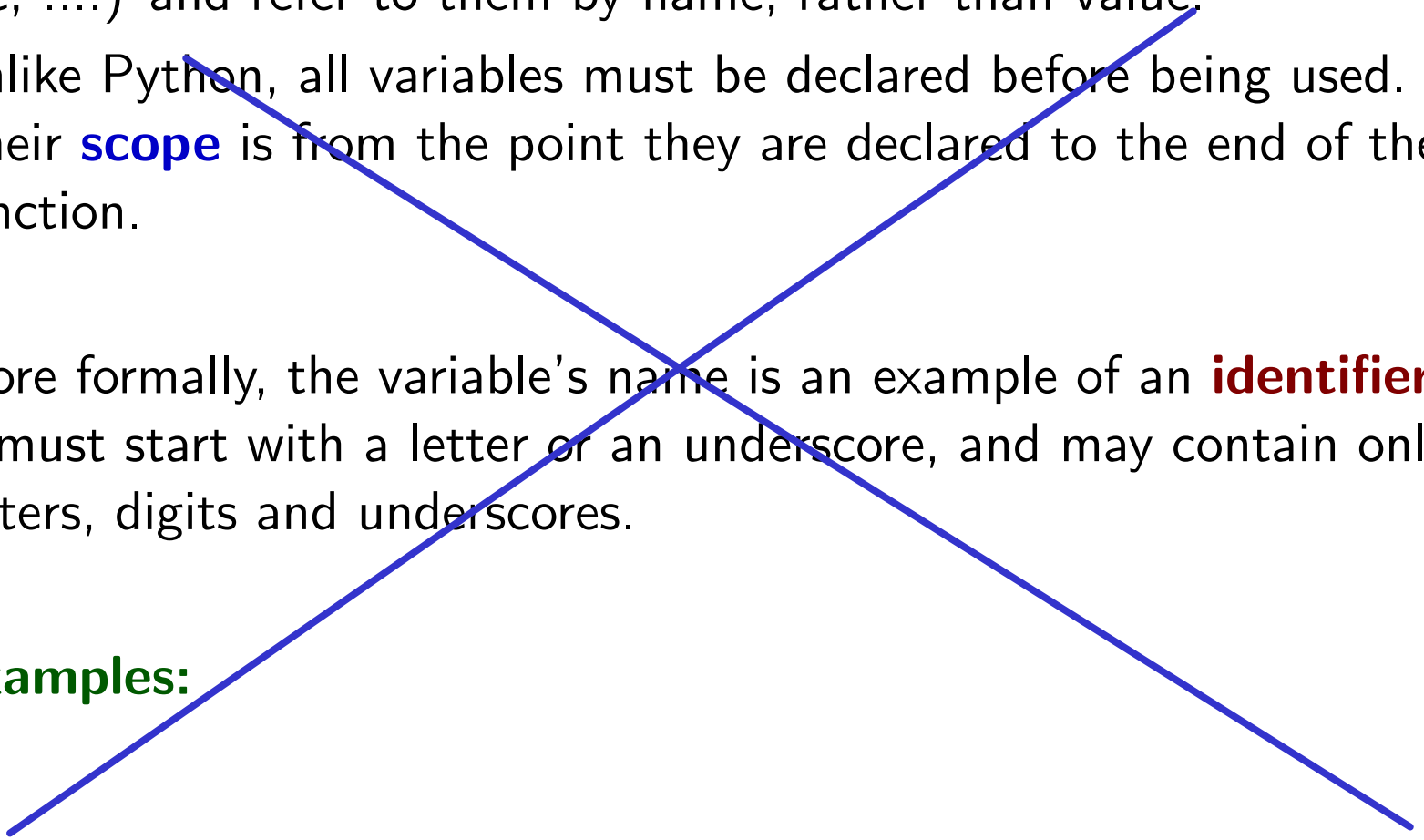
7. Variables

Variables are used to temporarily store values (numerical, text, etc, ...) and refer to them by name, rather than value.

Unlike Python, all variables must be declared before being used. Their **scope** is from the point they are declared to the end of the function.

More formally, the variable's name is an example of an **identifier**. It must start with a letter or an underscore, and may contain only letters, digits and underscores.

Examples:



7. Variables

All variables must be defined before they can be used. That means, we need to tell the compiler the variable's name and **type**.

Every variable should have a **type**; this tells use what sort of value will be stored in it. The type does not change (usually).

Python comparison

In Python, one "declares" a variable just by using it. The type of the variable is automatically determined. Furthermore, its type can change when we change the value stored in the Python variable.

This is one of the reasons why Python is so flexible, and so slow.

7. Variables

The variables/data types we can define include

- `int`
- `float`
- `double`
- `char`
- `bool`