# Programming and Data Analysis for Scientists

**C++ Workshop 1**

The shell and C++

**Prof Stephen Clark**

University of BRISTOL

SCIF20002

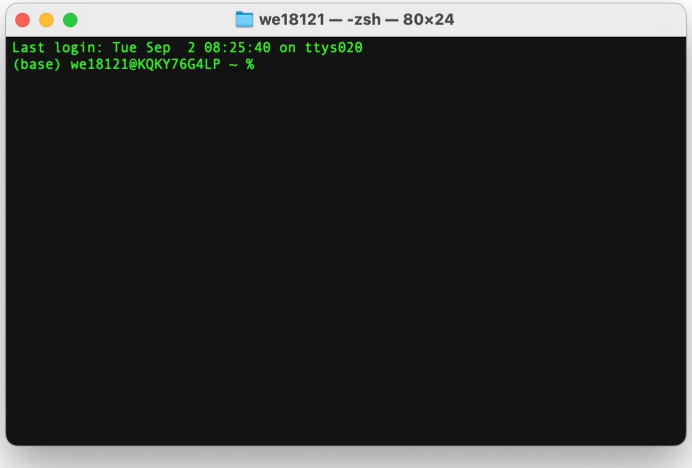
# The shell and C++

The purpose of this workshop is to introduce the C++ programming language. The *learning objectives* are:

- Learn some basic shell command line instructions
- Appreciate the advantages and uses of C++
- Understand what compilation of C++ code means
- Compile, run and post-process data from a simple C++ computation

# Using the **shell** terminal

Back in the 1970's a *terminal* was a physical device for logging into a mainframe. These days it refers to a minimal text input and output graphical application emulating this device.



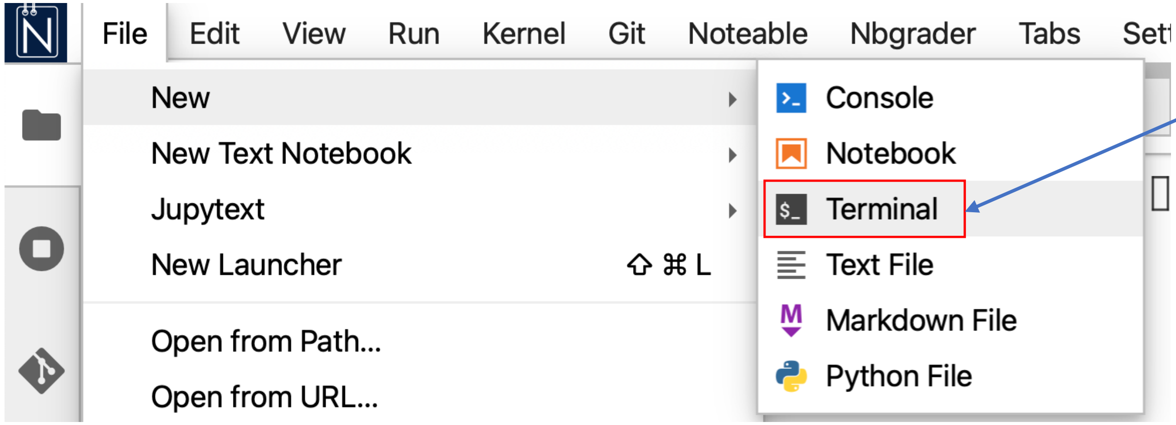
```
Last login: Tue Sep  2 08:25:40 on ttys020
(base) we18121@KQKY76G4LP ~ %
```

The *shell* is a command-line interpreter program. You type commands into the terminal, which sends them to the shell (like `Bash` or `Zsh`) for execution and the return results are displayed back in the terminal window.

While primitive in appearance the shell is extremely powerful. Any interactions with a high-performance computing facility will be via a shell.

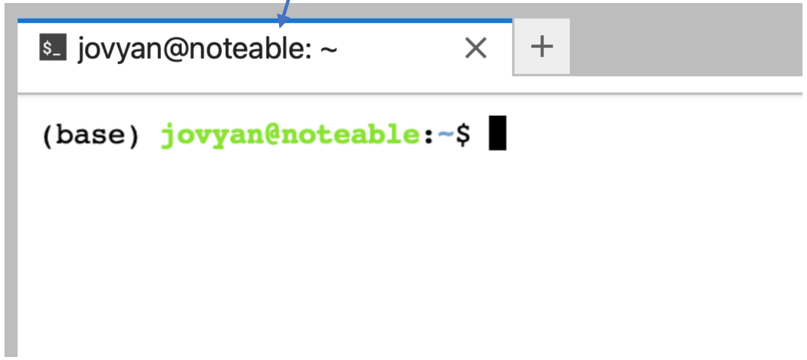
# Start a shell on ***Noteable***

You are already familiar with the ***Noteable*** server for running your Python Jupyter lab. The same resource can be used here. Start up Jupyter server …



Create Terminal session

It appears as a new tab in the Jupyter lab in your browser

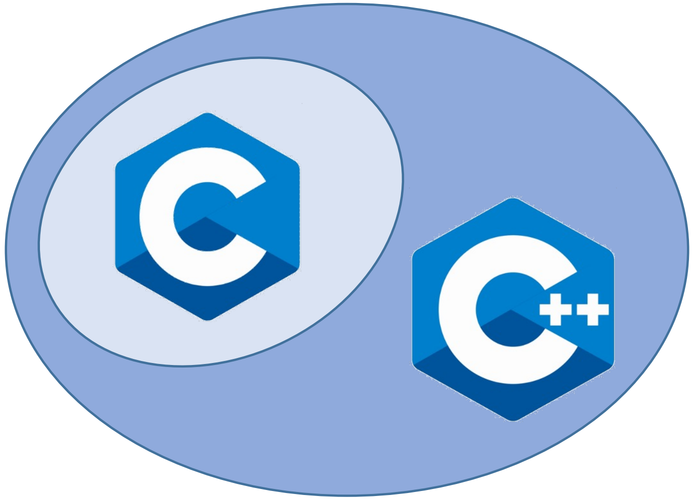


```
(base) jovyan@noteable:~$ █
```

We can run shell commands here that will be executed on the Noteable server.

# What are C and C++?

C is a procedural programming language developed in 1972. It is a light-weight mid to low-level language developed for systems programming.



C++ is a superset of C first developed in 1985 to enhance it to include object orientated formalism.

C++ uses C syntax along with many new extensions. It continues to be updated and evolve.

Elegant base syntax of C is the *lingua franca* of modern programming. Not only C++, but Java, C#,  as well as Python inherited many key features from it.

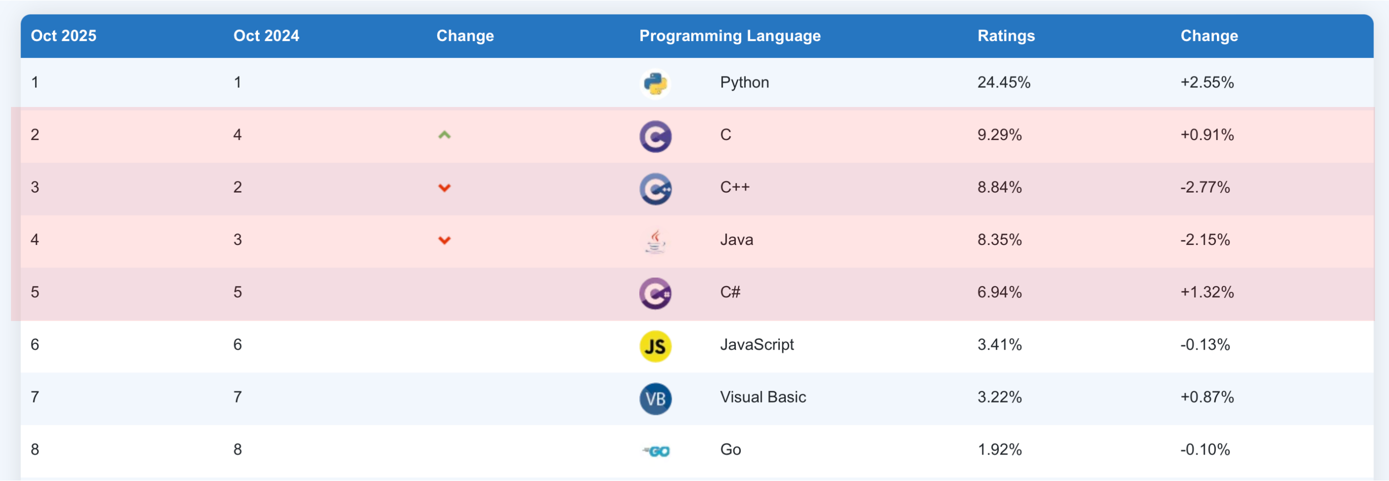
# Why bother to learn C++ in 2025?

Legacy, maturity and ecosystem. It has been around for over 40 years. Unix, Mac OS, Linux, Windows, Google’s Android OS are all written and continue to be developed in C++. Most problems have a C++ solution already written.

Since it is compiled it is **fast**! Most new computationally intensive applications like machine learning, scientific computing, virtual machines, device drivers, high-frequency trading, video games … are all written in C++.

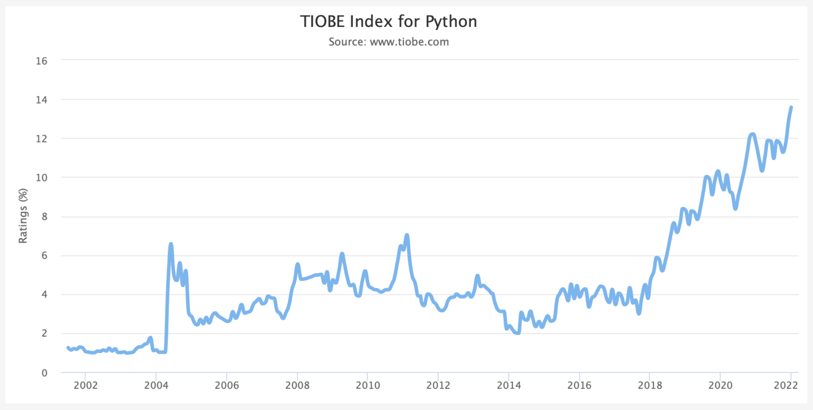
Since it is compiled it can be **small**! For embedded devices powering the IoT memory and processing is limited (due to power consumption and physical size) so their firmware is almost always written in C++.

How popular are C and C++ currently?



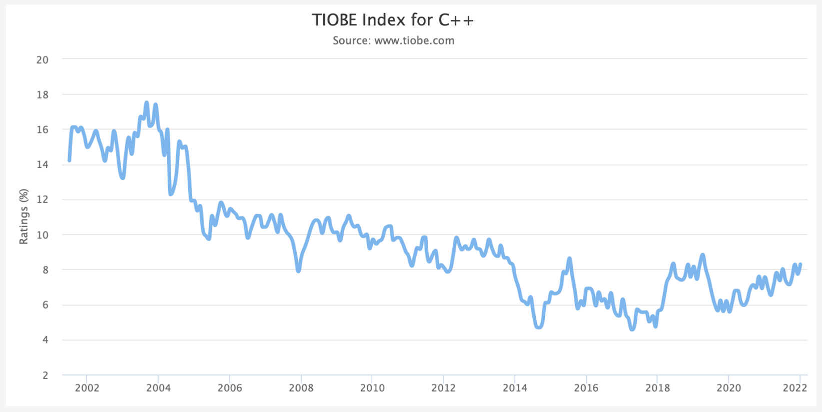
| Oct 2025 | Oct 2024 | Change | Programming Language | Ratings | Change |
|---|---|---|---|---|---|
| 1 | 1 | | Python | 24.45% | +2.55% |
| 2 | 4 | ▲ | C | 9.29% | +0.91% |
| 3 | 2 | ▼ | C++ | 8.84% | -2.77% |
| 4 | 3 | ▼ | Java | 8.35% | -2.15% |
| 5 | 5 | | C# | 6.94% | +1.32% |
| 6 | 6 | | JavaScript | 3.41% | -0.13% |
| 7 | 7 | | Visual Basic | 3.22% | +0.87% |
| 8 | 8 | | Go | 1.92% | -0.10% |

C and C++ still **very** well used.



TIOBE Index for Python

Source: www.tiobe.com



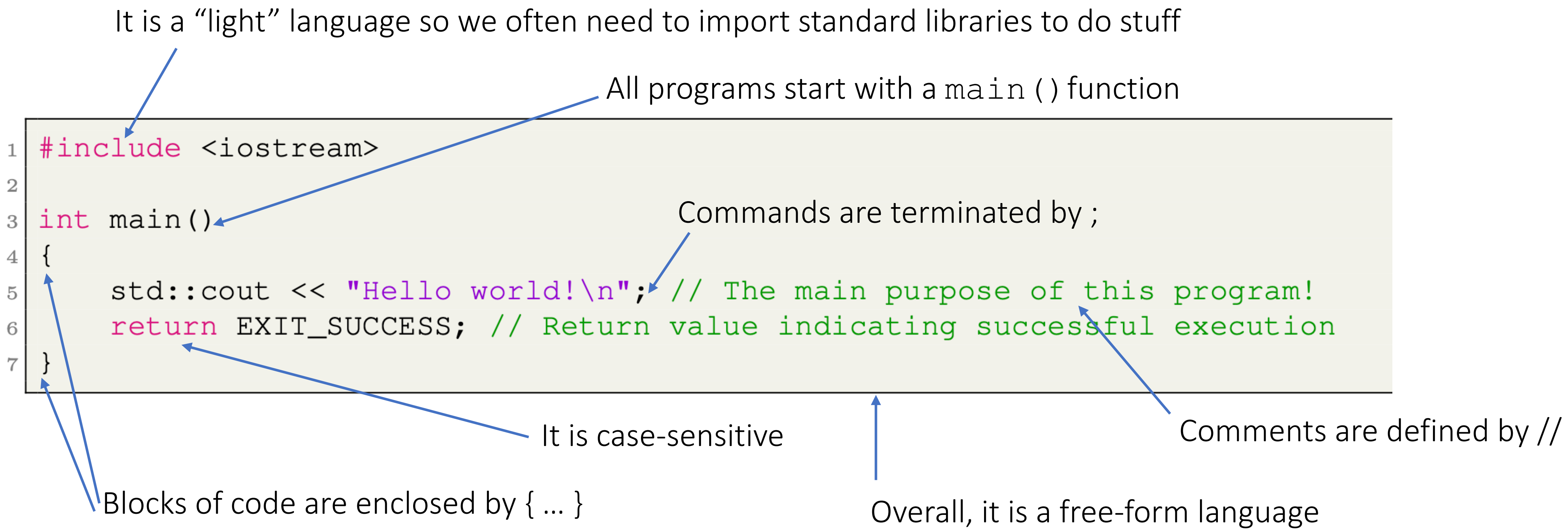
TIOBE Index for C++

Source: www.tiobe.com

Source: tiobe.com

# Anatomy of a C++ program

Take the customary first program anyone writes `hello.cpp`. We can highlight a number of essential features about C++ code:



It is a "light" language so we often need to import standard libraries to do stuff

All programs start with a `main()` function

```
#include <iostream>

int main()
{
    std::cout << "Hello world!\n"; // The main purpose of this program!
    return EXIT_SUCCESS; // Return value indicating successful execution
}
```

Commands are terminated by ;

It is case-sensitive

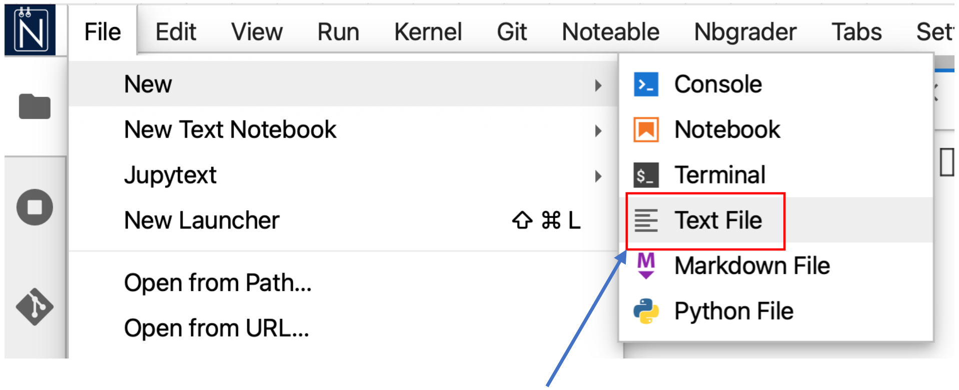
Comments are defined by //

Blocks of code are enclosed by { … }

Overall, it is a free-form language

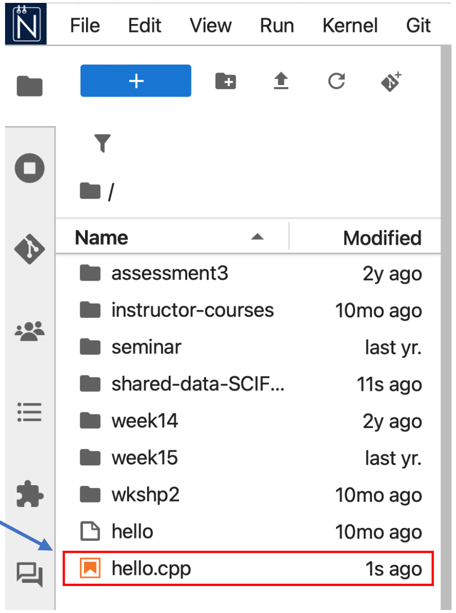
# Using *Noteable* for C++ code

C++ source files are just plain text files with a .cpp extension. We can open or create a file in Noteable and use the built-in code editor …

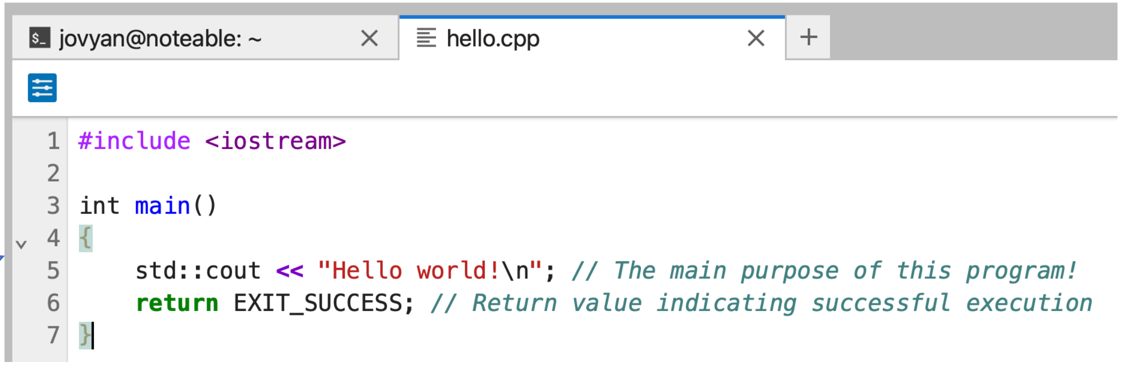


Create a new text file opening the editor

Open existing file in cloud filesystem



| Name | Modified |
|---|---|
| assessment3 | 2y ago |
| instructor-courses | 10mo ago |
| seminar | last yr. |
| shared-data-SCIF… | 11s ago |
| week14 | 2y ago |
| week15 | last yr. |
| wkshp2 | 10mo ago |
| hello | 10mo ago |
| hello.cpp | 1s ago |



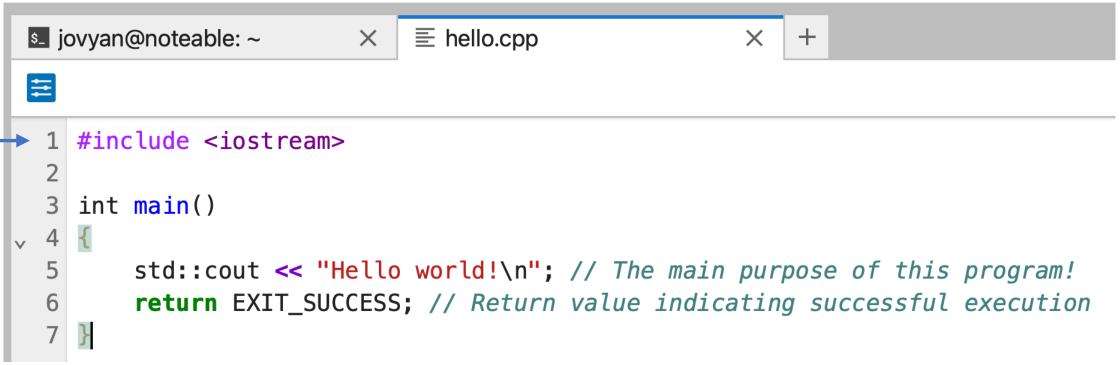
```
#include <iostream>

int main()
{
    std::cout << "Hello world!\n"; // The main purpose of this program!
    return EXIT_SUCCESS; // Return value indicating successful execution
}
```

New or existing file is opened as a new tab

# Running C++ code on *Noteable*

When editing a .cpp file C++ language syntax highlighting will automatically switch on. If you created a new text file then rename it <something>.cpp, and then highlighting will turn on:

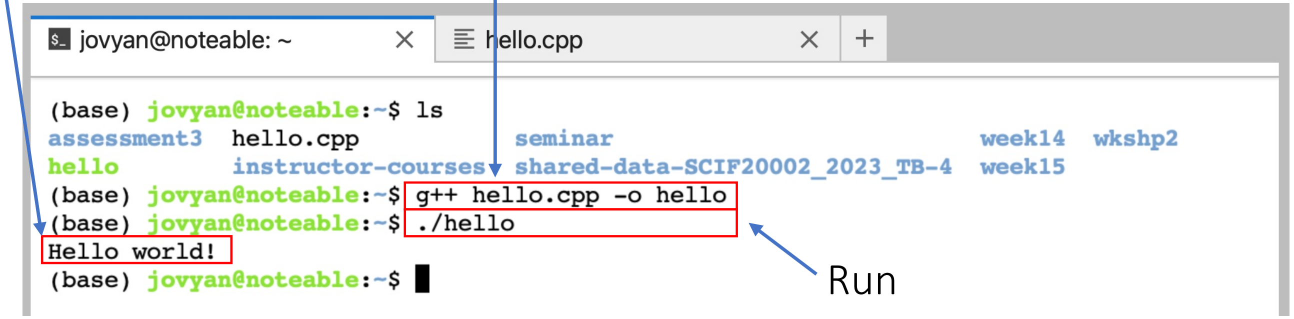
Language recognition



```cpp
#include <iostream>

int main()
{
    std::cout << "Hello world!\n"; // The main purpose of this program!
    return EXIT_SUCCESS; // Return value indicating successful execution
}
```

Output

Compile

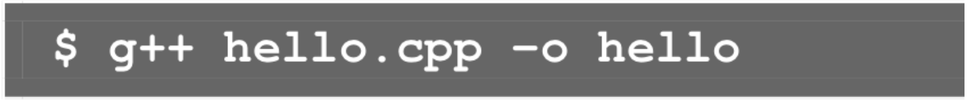


```
(base) jovyan@noteable:~$ ls
assessment3   hello.cpp          seminar                          week14  wkshp2
hello         instructor-courses shared-data-SCIF20002_2023_TB-4  week15
(base) jovyan@noteable:~$ g++ hello.cpp -o hello
(base) jovyan@noteable:~$ ./hello
Hello world!
(base) jovyan@noteable:~$
```

Run

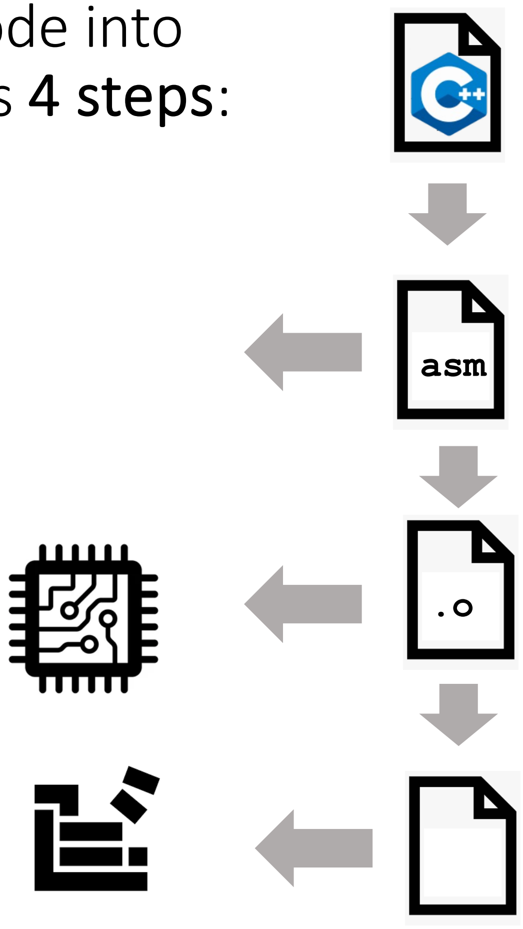
Switch to shell tab to compile and run in terminal

# What is compilation?



```
$ g++ hello.cpp -o hello
```

The process of converting source code into binary machine code. It typically has **4 steps**:

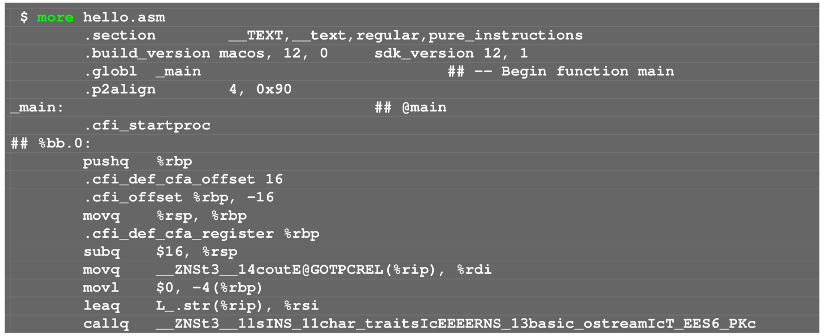


Preprocessing

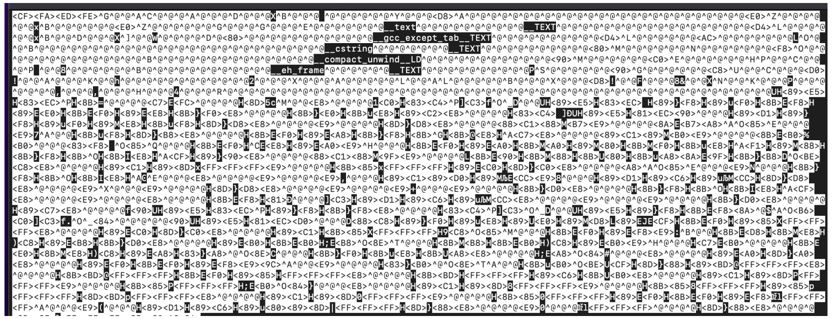
Compilation

Assembly

Linking



```
 $ more hello.asm
        .section        __TEXT,__text,regular,pure_instructions
        .build_version macos, 12, 0     sdk_version 12, 1
        .globl  _main                           ## -- Begin function main
        .p2align        4, 0x90
_main:                                  ## @main
        .cfi_startproc
## %bb.0:
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset %rbp, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register %rbp
        subq    $16, %rsp
        movq    __ZNSt3__14coutE@GOTPCREL(%rip), %rdi
        movl    $0, -4(%rbp)
        leaq    L_.str(%rip), %rsi
        callq   __ZNSt3__11sINS_11char_traitsIcEEEERNS_13basic_ostreamIcT_EES6_PKc
```



# What is HPC?

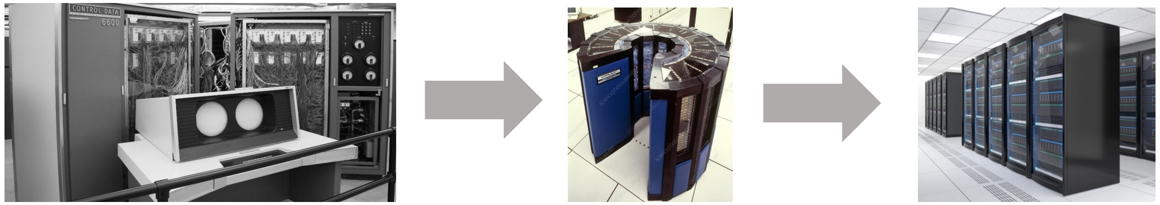
A “supercomputer” is a specially optimized computer that achieves high performance in floating-point operations (FLOPS) central to scientific calculations.

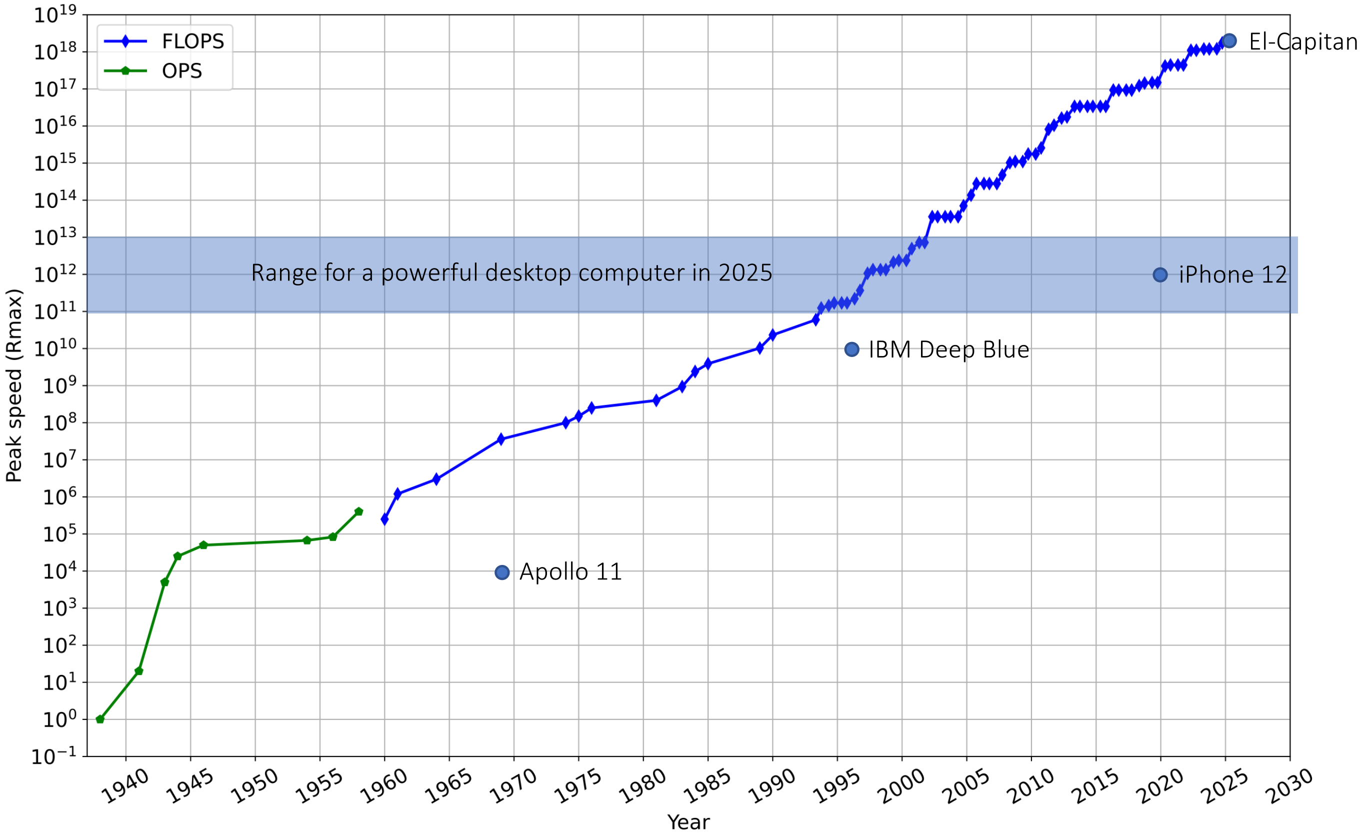
From the 1990’s onwards supercomputers evolved from being a dedicated single machine to being built from 10,000’s of networked “off-the-shelf” machines allowing for massive parallelization.

Since 2017 all the top 500 supercomputers run a Linux-based OS.

# What is HPC?

Currently (2025) the most powerful supercomputers can do in excess of $10^{18}$ FLOPS distributed over millions of CPU cores.





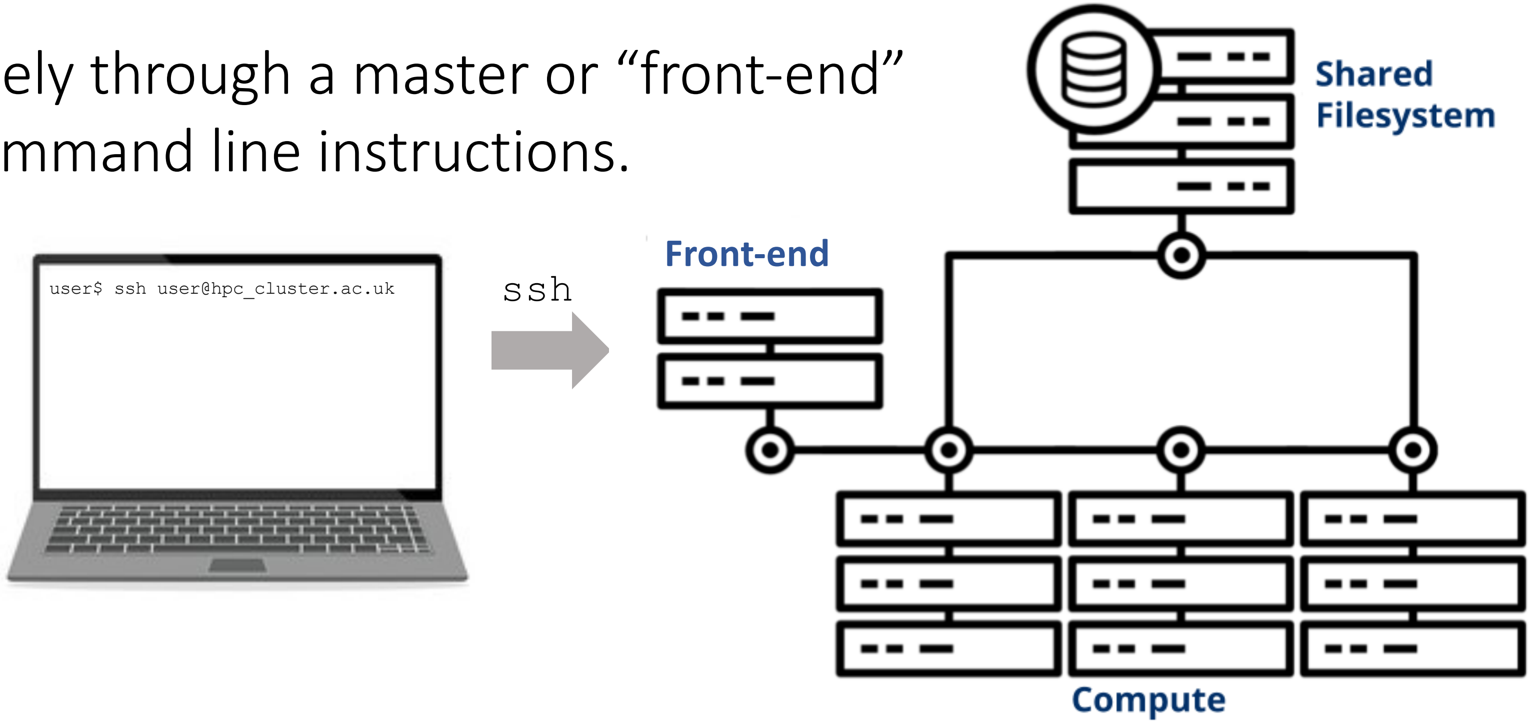
Source: Wikipedia/wiki/Supercomputer

# Anatomy of a HPC cluster

Modern HPC cluster comprise many high-powered compute nodes connected via a superfast internal network.

It is accessed remotely through a master or “front-end” node via **ssh** and command line instructions.

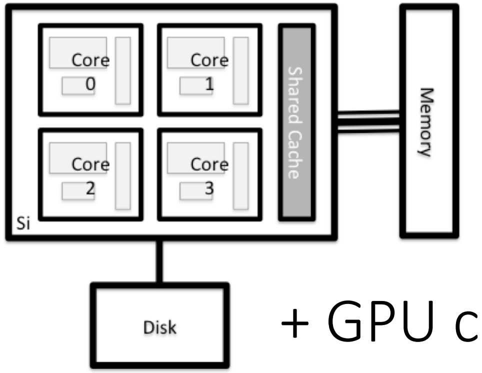
Code and data are stored on a shared filesystem.



```
user$ ssh user@hpc_cluster.ac.uk
```

# HPC compute nodes

A “compute node” in a HPC cluster is individually a very high-powered (often dual multicore (12-24) processors) large memory (4-10GB per core) computer packaged compactly as a **rack-mounted** server blade:



Compute node

+ GPU cards

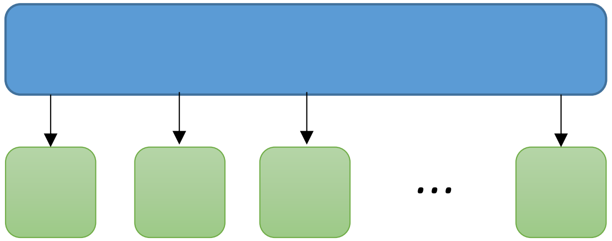
Server rack

Air-conditioned room

# Why do we need HPC?

→ See upcoming lecture

HPC enables complex scientific calculations that are **impossible** or **infeasibly slow** on a standalone computer. Two cases arise:

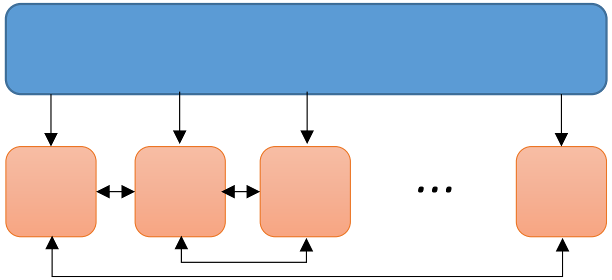


- *"Embarrassingly parallelizable problems"*
  Naturally divides into many independent tasks whose computation requires no communication between them.

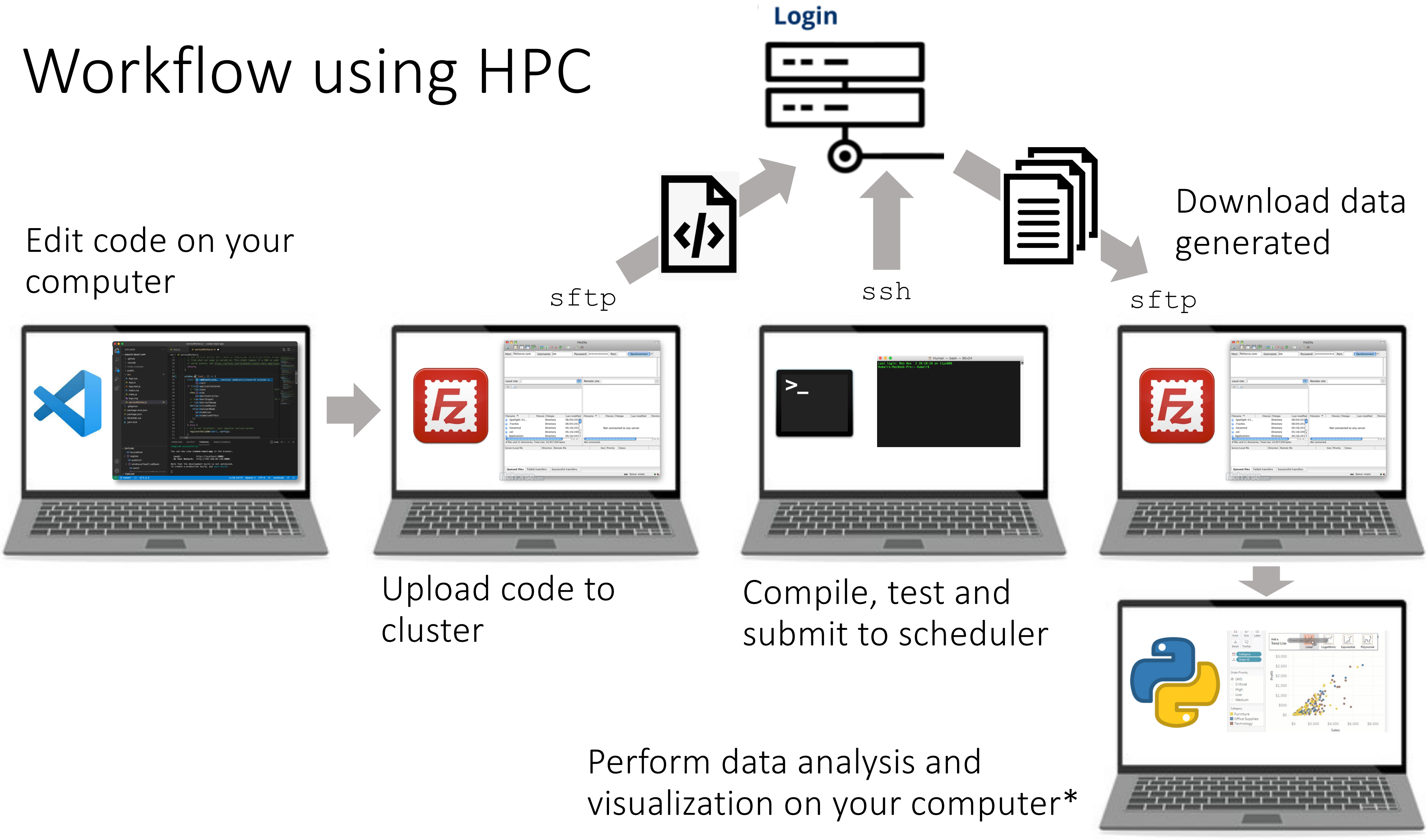
  Example: *Monte Carlo sampling*

- *"Complex parallelization problems"*
  Only divides into many dependent tasks whose computation requires substantial communication between them.

  Example: *High-resolution fluid dynamics*



# Workflow using HPC



Edit code on your computer

Upload code to cluster

Compile, test and submit to scheduler

Download data generated

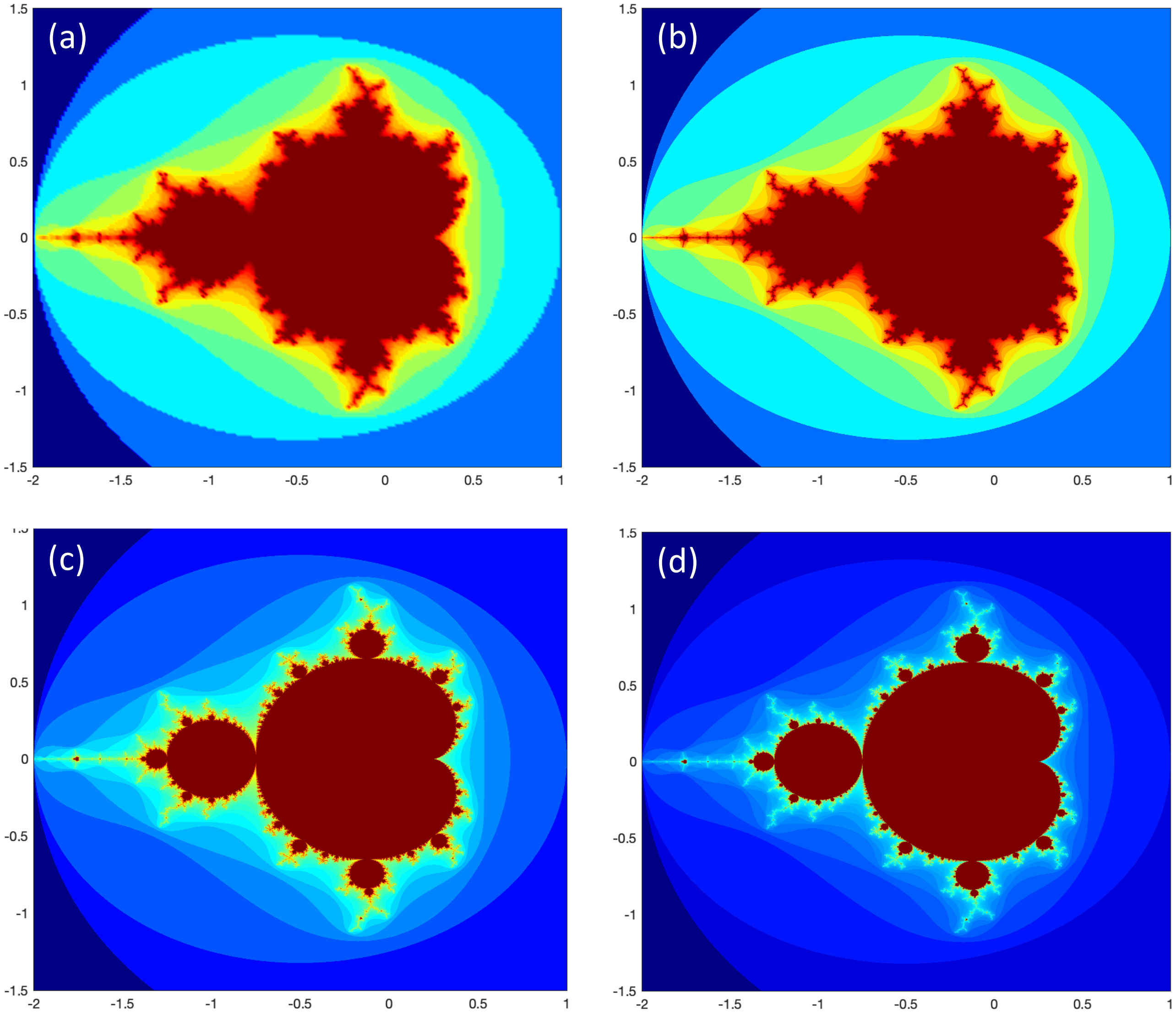
Perform data analysis and visualization on your computer*

```
Last login: Mon Nov  3 20:10:58 on ttys000
Humairs-MacBook-Pro:~ Humair$
```

# Neat example:

You are given C++ code that can compute the well-known Mandelbrot set fractal. The last task of this workshop is to compile it, run it and then post-process the output data to generate a plot similar to these.

An **optional extra** is to do very high-resolution calculation using HPC.


(a)
(b)
(c)
(d)
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-1.5
-1
-0.5
0
0.5
1