High Performance Libraries

- 01_Libraries for performance

  - Libraries are intended to help programmers in various ways
    - **Allow you to perform tasks faster than you could with the code that you'd write yourself** (this is especially important for Python programmers because a lot of Python libraries are written in C/C++/Fortran and can be orders of magnitude faster than Python code)
  - We're going to concentrate here on performance libraries which are intended to do a standard task faster than a normal programmer would be able or would have time to do
    - BLAS (Basic Linear Algebra Subprograms) is the canonical example of this but there are many, many libraries that fall into this area
    - Most libraries written in a compiled language and called from Python

Info:

We're providing you with a Fortran code that shows how much of a performance improvement you can get by using an optimised library. This is called **01_BLAS.f90**

Try:

- We've supplied a Fortran code with this example that multiplies together two n x n random matrices and times how long it takes.
  - Fortran is well suited for this because it has a built in matrix matrix multiply function **matmul**.
  - Many Fortran compilers have options that allow them to use the BLAS library to improve performance for **matmul** operations, including the free gfortran compiler
  - To run this you need the gfortran compiler and a BLAS library (we'd recommend OpenBLAS). You can install both through a package manager for most UNIX and UNIX-like OSes

- Compile the code with **gfortran -O3 01_BLAS.f90 -omatrix_timer**
- Then run the program with **./matrix_timer <n>** where **<n>** is the size of the matrices to multiply. Start with a number about 1000 and increase it until it takes about 30 seconds to complete (the number will depend on your computer, but will probably be less than 4000)
- Now compile it with **gfortran -O3 01_BLAS.f90 -omatrix_timer -fexternal-blas -lopenblas**
- Now run the code again with the same matrix size that you had before
- You should find that your code runs **much** faster. OpenBLAS is both substantially faster and also makes use of multiple cores which the default gfortran matmul function doesn't

- 02_Libraries for generality

- As well as just speeding up your code you can also use libraries to make your code more general
  - By general we mean a mixture of things
    - Better suited to different hardware platforms
    - More future proof
    - More extensible
- The does overlap with 01 somewhat because by being able to match the hardware that you are working on better will generally also improve performance
- Here we're going to demonstrate the value of this by switching from using a CPU based FFT library to using a GPU based FFT library

Info:

We're providing you with a Python code which uses Numpy's FFT function to blur an image. We will talk you through switching to using a GPU accelerated FFT algorithm using pyculib. You will need access to a machine with an nVidia GPU using a

Try:

- Make sure that you have the Numpy, Pyculib and Scipy libraries installed on your Python distribution. Both are available through pip

- We'll base this off the Notebook we saw earlier in the Core "Notebook To Script Module". We have provided a version of the notebook that has already been run through the conversion process. This is called **ImageBlur.py**. Run it and check that it produces a blurred penguin
- There are only three lines that do the actual FFT (two to transform the original image and the smoothing kernel and then one to reverse transform the blurred image)
- To move to using Pyculib you'll need to read the documentation for the FFT routines in that library. Try to find them using the search engine of your choice.
  - If you can't find them they are located at https://pyculib.readthedocs.io/en/latest/cufft.html
- You'll spot that the only difference is the import that is needed and the fact that rather than returning an FFTd version of the array you hand it it takes a second parameter to the fft and ifft calls
- One of the advantages of libraries is that the same library will be designed to work well on different systems
  - But this isn't always true if you're moving between different enough systems and you want the absolute best performance
- Try to replace the numpy.fft.fft and numpy.fft.ifft calls with equivalent Pyculib calls. An example of this working is in the Skills module on GPUs if you want some hints (in **Part2-Kernels**)
- If you are likely to want to switch between libraries then it can make sense to write your own routines (often called shim routines) that provide you with a consistent interface whatever underlying library you are using
  - Advanced : Try writing shim routines that allow you to switch between numpy and pyculib (possible extension, write it so that it uses pyculib unless it can't import it in which case it switches to using numpy)
- Advanced : Pyculib does tie you to working with nVidia GPUs. There is a library called reikna that works with many more GPUs (including AMD). Can you use reikna?