1 Exercise 01

All `program`s *must* contain `implicit none`. While not strictly necessary, it will eliminate an entire class of bugs.

You should use the lecture materials for help/inspiration, but please don't copy and paste! There is some value to be had in typing up the programs yourself.

There may be several ways to solve each problem. If you have time, you might like to try different approaches.

1.1 Hello World!

1. Open a text file called `hello.f90`. Write a simple "hello world" program that prints a message to the screen. Include at least one comment. Use `gfortran hello.f90` to compile it to `a.out`. Run your program with `./a.out`. Check it does what you think it should.
2. Time to break the program! Delete the `p` in `program` and try to recompile. What happens? What does the error message say?
3. Undo the deletion. Delete another character and see if it breaks the program, and if so, how. How many unique error messages can you find from single-character deletions? Which characters don't matter?

1.2 Hello <name>!

1. Write up the `hello_input` program from the lectures into a new file, `hello_input.f90`. Compile it, this time using the `-o` flag to give the executable a name. Run the program and give it some input.
2. Try the following ways to break the program. For each method, try to explain why the program behaves the way it does.
    1. Enter two words when it asks your name
    2. Enter a single word longer than 20 characters
    3. Enter a number with a decimal point
3. Create a second `character(len=20)` variable, try reading the two `character` variables with a single `read`, and add your new variable to the `print`

1.3 Summing integers

1. Write a program that sums all the integers from 1 to 100
2. Modify the program so that it takes an integer from user input, and then sums all the integers up to that number.

1.3.1 Further

1. What happens if the user supplies a negative number? Hint: try looping over the `read` until the number is acceptable

1.4 Solving quadratics

The solutions to the quadratic $ax^2 + bx + c = 0$ is

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

1. Take three real numbers from user input, and print the two solutions for x using the above formula. If there are no real solutions, print a message saying so.
2. Extend your program to also print complex solutions

1.5 Reversed Range

1. Write a program that creates an array filled with the numbers 1 to 100 and prints it to screen.
2. Read two integers in the range $[1, 100]$ from the user and print out the array within that range.
3. Using the two numbers from step 2, reverse the numbers in the array within that range, and print the entire array.

1.5.1 Further

1. What happens if the user gives numbers outside the range $[1, 100]$? What are some ways of handling this?
2. What happens if the user gives the two numbers in the opposite order than you're expecting? What are some ways of handling this?

1.6 Factorial

The factorial of n is

$$n! = 1 \cdot 2 \cdot 3 \cdots (n-2) \cdot (n-1) \cdot n.$$

1. Write a function that computes $n!$ by using a `do` loop
2. Write a `recursive` function that computes $n!$ by calling itself with $(n-1)$.
3. Write a function that can compute $n!$ on every element of an array

1.7 Cross product

The cross product of two three dimensional vectors is given by:

$$a \times b = (a_2 \cdot b_3 - a_3 \cdot b_2)\hat{\mathbf{i}} + (a_3 \cdot b_1 - a_1 \cdot b_3)\hat{\mathbf{j}} + (a_1 \cdot b_2 - a_2 \cdot b_1)\hat{\mathbf{k}}$$

1. Write a subroutine that returns the cross product of two arrays via an out argument.
2. Rewrite your subroutine as a function that returns the cross product in the result.

1.8 Mean and standard deviation

The mean, $\bar{x}$, and standard deviation, σ, of a set x of n numbers are given by

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x(i)$$

and

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x(i) - \bar{x})^2}.$$

1. Write a subroutine that takes an array and returns its mean and standard deviation via out arguments.
2. Given a `real` scalar or array `r`, `call random_number(r)` will fill `r` with a random number between 0 and 1. You can use this to verify your subroutine is working correctly: the mean of a uniformly distributed set of numbers between $[0, 1)$ is 0.5, and the standard deviation is about 0.29.

1.9 Euler Integration

The Euler update formula for some function $f(y, t)$ is

$$y^{n+1} = y^n + (\Delta t)f(y^n, t^n),$$

where $y^n = y(n\Delta t)$ is an approximation to $y(t)$, Δt is the timestep, and t^n is the current time. Use this formula to solve the ODE

$$f(y, t) = \frac{dy}{dt} = \sin^2(t), y(0) = 0.$$

The exact solution at $t = \pi/2$ is $y = \pi/4$.

1. Write a program that has two functions: one that returns $f(t) = \sin^2(t)$, and one that returns y^{n+1} given y^n, Δt and t^n. The second function should call the first. Use the default `real` for floating point variables.
2. Read an integer N from the user, and then take N timesteps from 0 to $\pi/2$. Compute the error. How does it vary as you increase N?
3. Change the `real` variables to kind `real64`. Now what is the error as you increase N? Make the kind a parameter so that you can change between `real64` and `real32` and see the difference.

1.10 Calculating π

It is possible to use the `random_number` routine from above to calculate π. The method is as follows:

1. Generate a random point in a 2D surface between $[-1, 1]$. You will need to generate two numbers here, one for x and one for y.
2. Calculate the distance from the origin $(0, 0)$. (Hint: the intrinsic `hypot` may be useful here).
3. Repeat steps 1. and 2. for a total of N points.
4. Determine how many points are less than a distance of 1 from the origin. Using the ratio of the number of these points to the total N, along with $A = \pi r^2$, calculate π.
5. Repeat the above steps for varying N until the answer has converged in N.