- Introduction to Object Oriented Programming
- Classes, Objects and Methods

Additional material that you may find useful:

- Magic Methods and Operator Overloading

1.5 Marking Scheme

- Polynomial Checkpoint Marking Scheme

2 Radioactive Decay

2.1 Aim

The aim of this checkpoint is to write an application to simulate the radioactive decay of unstable nuclei.

Radioactive decay is a statistical process; it is impossible to predict when a given nucleus will decay, only a probability that it might. As the decay of individual nuclei are random events, if there are N nuclei present at time t, the number of nuclei ΔN which decay in a small time interval between t and $t + \Delta t$ is proportional to N:

$$\Delta N = -\lambda N \Delta t$$

where the constant of proportionality λ is the *decay constant*. ← EXPONENTIAL DECAY

We can also rearrange this expression to obtain the probability p that a nucleus decays within a given time interval Δt:

$$p = \frac{\Delta N}{N} = -\lambda \Delta t$$

(the minus sign simply means that the number of undecayed nuclei decreases).

Note that in the expressions above we have assumed that N is constant (to a good approximation) over the time interval Δt, i.e. that only a small fraction of the nuclei will decay during this time. This means that the probability of decay in the time interval Δt should be small, i.e. p should be much less than 1. In turn, this means that the time interval Δt should be short compared to the average lifetime τ, where $\tau = 1/\lambda$. ← AVERAGE LIFETIME, H

2.2 Checkpoint task

Iodine-128 is a radionucleide often used as a medical tracer. It has a half-life $T_{1/2} = 24.98$ minutes, giving a decay constant $\lambda = 0.02775$ min^{-1}.

Your task is to write a PYTHONapplication to simulate the radioactive decay of a 2D array (or list) of $N \times N$ Iodine-128 nuclei. (Note there is nothing significant in a 2D array here. It is just a convenient way to 'hold' and visualise the atoms.)

Your code should prompt the user for the value of the decay constant λ, the length of the 2D array N and the timestep Δt. For Iodine-128, suggested values are $N = 50$ and $\Delta t = 0.01$ min.

Your code should then simulate radiative decay by randomly selecting undecayed nuclei to decay as follows: At each timestep it should iterate over all of the elements in the array, check each to see whether it has decayed and if not, determine whether or not to decay it by generating a random number and comparing it to p, the probability of decay in the given time interval.

Rather than run until all of the nuclei have decayed (which could take many iterations - remember that decay is a random process!), your code should terminate when the number of undecayed nuclei has fallen to half of the initial value. The time taken to reach this point is the 'simulated' value of the half-life $T_{1/2}$. It should then print out:

- A visual representation of the array of nuclei; this should clearly distinguish between undecayed and decayed nuclei, for example by using 0 and 1 for decayed and undecayed nuclei respectively.
- The initial and final number of undecayed nuclei.
- The simulated value of the half-life.
- The actual value of the half-life, as given by the decay constant.

Before attempting to code the problem you should write an object-oriented design for your code. This should show what classes you need together with the class and/or instance variables and methods associated with each class. It is worth remembering that although there is no single correct way of designing the code, some designs are better than others and a good program design will make your code easier to write, read and debug. A design is particularly important here if you go on to consider the optional extras (below). You should at least consider these in your design, even if you do not implement them.
Ask a demonstrator or a friend to comment on your code design before starting to write your code. Update your design if necessary.

2.3 Optional extras

Estimate of uncertainty

Modify your code to obtain an estimate of the uncertainty in your simulated half-life. (You will need to think how to do this.) Is the result consistent with the actual value of the half-life? If so, you can claim to have verified the code for this problem.

Decay series

For heavy nuclei, a daughter nucleus resulting from radioactive decay, may itself be unstable and undergo decay. The process may continue through several "generations" until the decay results in a stable nucleus. This is called a decay series or decay chain.

There are four radioactive decay series found in nature: the decay of Uranium-235, Uranium-238, Thorium-232 and Neptunium-237. ← NATURAL DECAY SERI

Modify your code to simulate a radioactive decay series which decays through two or more generations. Each element in the series should have a different half-life. You can either choose your own values for a hypothetical series, or select values from a section of one of the natural decay series; for example, the half-lives for the last four decays in the Uranium-235 series are:

Lead-211 $\overset{T_{1/2}=36\,\text{mins}}{\longrightarrow}$ Bismuth-211 $\overset{T_{1/2}=2.1\,\text{mins}}{\longrightarrow}$ Thallium-207 $\overset{T_{1/2}=4.8\,\text{mins}}{\longrightarrow}$ Lead-207 (stable)

2.4 Relevant course sections

Studying the following course sections will help you complete this checkpoint:

- Classes, Objects and Methods
- Object Oriented Design

Additional material that you may find useful:

- Errors and Exceptions

2.5 Marking Scheme

- Radioactive Decay Checkpoint Marking Scheme

3 Mandelbrot Set

"Clouds are not spheres, mountains are not cones, coastlines are not circles, and bark is not smooth, nor does lightening travel in a straight line."

Benoit Mandelbrot