1 Polynomial

1.1 Aim

The aim of this checkpoint to write a PYTHONclass to represent a polynomial and to use this class to perform simple mathematical operations on polynomials.

1.2 Checkpoint task

Write a PYTHONclass to represent a polynomial of the form:

$$P(x) = \sum_{i=0}^{n} a_i x^i = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

where the coefficients a_i are real numbers.
Your class should include:

- A `list` instance variable to hold the coefficients of $P(x)$.
- An `__init__()` method to set the values of the coefficients a_i, which should be supplied as a `list`.
- Methods to:
  - Calculate and return the order of $P(x)$.
  - Add another polynomial to $P(x)$ and return the result as a new polynomial. Your code should include the case where the polynomials being added are of different order. ← ADDING POLYNOMIALS
  - Calculate the derivative of $P(x)$ and return the result as a new polynomial. ← DIFFERENTIATING POL
  - Calculate the antiderivative (indefinite integral) of $P(x)$ and return the result as a new polynomial. A numerical value for the constant of integration should be supplied to the method. ← INTEGRATING POLYNO
  - Print a sensible `String` representation of $P(x)$, for example in the form:

```
P(x) = a0 + a1x + a2x^2 + .... + anx^n
```

You should also write a test code to check that your class works. This should be a separate file and have a `main()` method that creates the following polynomials:

- $P_a(x) = 2 + 4x^2 - x^3 + 6x^5$
- $P_b(x) = -1 - 3x + 4.5x^3$

and then:

- Calculates the order of $P_a(x)$
- Adds $P_b(x)$ to $P_a(x)$
- Calculates the first derivative of $P_a(x)$
- Calculates the antiderivative of this result, i.e. the antiderivative of $dP_a(x)/dx$. The constant of integration c should be set to $c = 2$.

In each case your code should print the results to the screen, using your `String` representation of a polynomial where appropriate.

Note: As this checkpoint is concerned primarily with writing and using classes, you may 'hard code' the values of the coefficients and the constant of integration in your test code; you do not need to input them from the terminal (or read them in from a file).

1.3 Optional extra

If you have the time, here is another exercise that employs basic OO functionality: write a class to generate (pseudo-)random numbers, given a suitable 'seed'.

A simple algorithm you can use to generate random numbers is :

```
seed = float(seed * a + c ) % m
randomNumber = abs(seed / m)
```

where `m`=233280, `a`=9301, `c`=49297 and `seed` is an integer.

Write a test class that prints 10 pseudo-random integers (between 0 and a specified maximum value) to the screen, using the methods in your random number generator class.

A common way of seeding the RNG is to use an integer such as your birthdate. This will produce the same list of 10 random numbers each time it is run. Why produce random numbers that are not random? Well, sometimes this is useful as a debugging tool or a way of testing the influence of changing only one parameter in a "randomly" initialised simulation.

Alternatively, more "randomness" can be inserted using the current time cast as an integer. Importing the module `time` and calling the the `time.time()` method (which returns the current time as floating point seconds since the 'epoch' - 12:00am, January 1, 1970) will enable you to do this. This method should not produce the same list of random numbers on subsequent runs.

At the end of all this, you may be dismayed (or relieved) to note that PYTHONhas built-in functionality for random number generation that is a lot more random than this simple algorithm : the method `random()` in the module `random`. ← MORE RANDOM NUMB

1.4 Relevant course sections

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

- 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 I

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, HA

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}.