Computer Modelling Exercise 2

A Particle3D Class

Due: 16:00 Thursday, Week 7, Semester 1

1 Aims

In this exercise, you will put into practice concepts of object-oriented programming by writing a new *class* that describes point particles moving in 3D space.

This exercise is different from anything you have seen in *"Scientific Programming"*. You will code a set of methods (class functions) with a *fixed and exact* specification. A successful exercise requires code that works *and* follows the design requirements.

The class you write will be reused for Exercise 3 and the project. It is an important stepping stone, and, after you receive feedback, you should implement the recommended changes back into the class.

2 Preparation

1. Read the overview of object-oriented programming (OOP) in the course notes, CompSim notes, and/or the OOP presentation, all accessible through Learn.
2. Make a new directory on your machine for this exercise.
3. Download the three files for the course from from Learn, and put them in your new directory:
    - `particle3D.py` a template you can start from for the code.
    - `test_particle3D.py` a tester code checking if your code is working.
    - `p3d_test_data.txt` a data file used by the test code.

4. Run the test program, `test_particle3D.py`. As in exercise one, you should see lots of failures, because the code is not written yet.

5. As you complete the tasks below, run the test code and check the feedback. If everything is working then the tests should start to pass. Again, the test code does not check everything, so don't expect full marks just because they all pass!

3 Understanding our class

3.1 Overview

Your class will collect the properties of classical point particles in 3D space, and functions (*methods*) which use or affect those properties. A single *instance* of the class will represent one particle.

If we have an instance of a python class in a variable called, for example, `p`, then we can access both the properties and methods using a dot, for example: `p.mass` or `p.kinetic_energy()`.

Standard methods in a class have the first argument `self`, meaning "this instance". Inside those methods we can then get attributes using `self.mass` or `self.kinetic_energy()`.

3.2 Particle Attributes

The properties of our particles are also called *attributes* in python. In this case they are:

- `label`, a string;
- `mass`, a float;
- `position`, as a `NumPy` float array;
- `velocity`, as a `NumPy` float array.

It is important that the position and velocity are `NumPy` arrays of the correct shape, (3), so we can take advantage of `NumPy` later on.

3.3 Instance Methods

After defining a class we next write a series of *instance methods*.

An instance method will operate on, or return a quantity, relevant to a single instance of the class. For example, in our case, an instance method could return the kinetic energy of a given particle, or its linear momentum. Another instance method would modify its position or velocity given the current total force acting on it.

In Python, instance methods are always written with `self` as the first argument:

```
def momentum(self):
    ...

def update_position_1st(self, dt):
    ...
```

Here, `self` represents *the specific instance of the class.* For example, if we had a variable `p` as an instance and then called `p.momentum()`, then inside the function `self` would be the same object as `p`. For example:

```
class Dog:
    def __init__(self, name):
        self.name = name
    def bark(self):
        print(self.name + " says woof")

# two different dogs represented by two
# different instances of the Dog class
rex = Dog("Rex")
rover = Dog("Rover")

rex.bark() # Here the "self" in the bark function means rex
rover.bark()  # Now it means rover instead
```

3.4 Magic Methods

"Magic methods" are another important constituents of a class. These are methods with additional functionality, whose special name is enclosed by "`__`".

The `__init__` method is particularly important in python: it is used to create new instances of a class. In our other code we create new instances using the name of the class. Here's an example of creating a particle at the origin with unit mass:

```
    p = Particle3D("A", 1.0, [0.0, 0.0, 0.0], [0.0, 0.0,
       0.0])
```

This code calls the magic `__init__` method under the hood - all the arguments here are passed to that method. Typically, the `__init__` method sets the attributes of the instance using these e.g. `self.label = label`.

The `__str__` method is another important magic method. Whenever we convert a variable to a string, either creating a new variable with `s = str(p)` or printing it out using `print(p)`, python runs `__str__` to decide what to print out. In this class we will use a specific format for this string, which will enable us to easily print the particle out in a format called XYZ used in particle animations.

3.5 Static Methods

There are quantities we want to compute, or methods to update the system, that are clearly related to particles in 3D space, but not tied to a particular particle. An example would be the centre of mass of our system, or total kinetic energy. These methods are implemented in *"static methods"*.

In Python, as static methods are not tied to a particular instance, these do not have `self` as an argument. However, we have to highlight these methods to Python using *"decorator"*, beginning with an @ sign:

```
@staticmethod
def total_momentum(particle_list):
    ...
```

Another important static method we will use here is one that *creates* a list of particles, from an input text file.

4 Tasks

Read and understand section 3 before jumping straight to this section.

Then fill in the missing content of the Particle3D class, as listed here.

4.1 Magic Methods

Task 1 Complete the two magic methods of the class:

- The `__init__` method that initialises the particle properties described in section 3.4. Your method should convert its inputs to the correct types. You will need to do this method first before you can test the others.
- A `__str__` method that returns a string describing the particle in an XYZ-compatible format[1]:

  ```
  <label> <x-pos> <y-pos> <z-pos>
  ```

4.2 Instance Methods

Task 2 Complete the instance methods of the class:

- A method `kinetic_energy` to return an object's kinetic energy.
- A method `momentum` to return an object's linear momentum.
- A method `update_velocity` to update the velocity of a particle from a current time t to the next time $t + \delta t$, given a time-step dt and force $\boldsymbol{f}(t)$, following

$$\boldsymbol{v}(t + dt) = \boldsymbol{v}(t) + dt \cdot \boldsymbol{f}(t)/m$$

- A method `update_position_1st` for a first-order update of the particle position for a timestep, following

$$\boldsymbol{r}(t + dt) = \boldsymbol{r}(t) + dt \cdot \boldsymbol{v}(t)$$

- A method `update_position_2nd` for a second-order update of the particle position for given timestep and force, following

$$\boldsymbol{r}(t + dt) = \boldsymbol{r}(t) + dt \cdot \boldsymbol{v}(t) + dt^2 \cdot \boldsymbol{f}(t)/2m$$

4.3 Static Methods

Task 3 Complete the static methods of the class:

[1]Without the <>, and with no extra puncatuation like commas

- A method `total_kinetic_energy` that returns the total kinetic energy of a list of particles
- A method `com_velocity` that returns the center-of-mass velocity of a list of particles.
- A method `read_line` that creates a particle from a line of text. It should take a string as argument and return a `Particle3D` object. The file `p3d_text_data.txt` contains examples of the line format, and the test program will read it.

It is a good idea to call methods *from* inside other methods. Re-using code like this helps reduce the number of possible mistakes.

4.4 Docstrings

Task 4 Ensure all your methods have correct docstrings, describing their purpose, inputs (if any) and outputs (if any).

5 Submission

Include your name or student number (but **NOT** the exam number) on the `particle3D.py` header. Avoid other personally identifiable data. Then submit the Python file through the course LEARN page, by **16:00 on Thursday, week 7 of semester 1**.

You must not change the method or attribute names.

Start working on the next exercise as soon as you are happy with your class and all the tests pass.

6 Marking Scheme

This assignment counts for 5% of your total course mark.

- Initialization and properties [1]
- Instance methods are correct [4]
- Static methods are correct [2]

- The `__str__` method is XYZ–compliant [1]
- Layout, docstrings, and comments [2]

Total: 10 marks.