Vector
Problem ID: a18p01vector

Write the class Vector that represents a three dimensional vector. It should be initialized with three numbers, one for each coordinate, and they should each have a default value of 0. We will leave it to you to decide how you want to store these values internally.

Below are the specifications of the expected behaviour of the class. We recommend that rather than reading through it all at the start, you should just start implementing one item at a time, make sure you pass the unit tests for that particular functionality, and then go on to the next one. The test groups are ordered this way, so you'll gradually be able to score higher and higher, as you implement more and more of the functionality.

You should only submit the class code.

The class should implement the following functionality:

1. Conversion to a string should work as shown in the following example:

   ```
   >>> v = Vector(12, 3, 4)
   >>> print(v)
   [[ 12 3 4 ]]
   ```

   This is a non-standard way to represent vectors of course, but one that suffices for our purposes in a command line interface, clearly distinguishing the representation from a python list or tuple for example.

   For this, you will need to implement the "`__str__()`" method.

   Remember that the "`__str__()`" method should never print anything, only return a string.

2. A 3-dimensional Vector has a length that can be calculated using the Pythagorean theorem. It might be tempting to use the `len()` function, which calls the "`__len__()`" method, to find the length, but Python insists that the `len()` function return an integer, and not all vectors have integer lengths. So instead we can use the `abs()` function. Finding the length of a Vector should then work as follows:

   ```
   >>> v = Vector(12, 3, 4)
   >>> abs(v)
   13.0
   ```

   For this, you will need to implement the "`__abs__()`" method.

3. Adding two Vectors should return a new Vector, whose first coordinate is the sum of the first coordinates of the two Vectors, and likewise for the other two coordinates, as demonstrated below:

   ```
   >>> u = Vector(12, 3, 4)
   >>> v = Vector(0, 10, -14)
   >>> w = u + v
   >>> type(w) == Vector
   True
   >>> print(w)
   [[ 12 13 -10 ]]
   ```

   For this, you will need to overload the + operator by implementing the "`__add__()`" method.

4. The class should also implement a subtraction operation in a similar way, overloading the − operator:

   ```
   >>> u = Vector(12, 3, 4)
   >>> v = Vector(0, 10, -14)
   >>> w = u - v
   >>> type(w) == Vector
   True
   >>> print(w)
   [[ 12 -7 18 ]]
   ```

For this, you will need to implement the "`__sub__()`" method.

5. Comparing two Vectors for equality should work as follows:

```
>>> u = Vector(12, 3, 4)
>>> v = Vector(12, 4, 3)
>>> u == v
False
>>> w = Vector(12, 3, 4)
>>> u == w
True
```

   For this, you need to implement the "`__eq__()`" method.

   Furthermore, since floating point operations can introduce a little error, such as when subtracting 1.0 from 1.2 results in $0.199\,999\,999\,999\,999\,96$ rather than 0.2, due to the way floating point numbers are represented in binary, comparison for equality should allow for a small error.

   A convenient way to do this is to subtract one Vector from the other, and check if the length of the resulting Vector is "close enough" to 0. For this we can use the `abs()` function, since we have already implemented the "`__sub__()`" method.

   Here, "close enough" means within a small tolerance, say $0.000\,000\,1$. You should use a constant for this tolerance.

6. (a) The class should also support the $\star$ operator, intended to be used for scaling the vector, i.e. multiplying it with a scalar (number). The result is a new Vector, where the first coordinate is the product of the scalar and the first coordinate of the original Vector instance, and likewise for the other two coordinates:

```
>>> v = Vector(12, 3, 4)
>>> w = v * 5
>>> type(w) == Vector
True
>>> print(w)
[[ 60 15 20 ]]
```

   For this, you will need to implement the "`__mul__()`" method.

   (b) It should also be possible to scale the vector from the left side:

```
>>> v = Vector(12, 3, 4)
>>> w = 5 * v
>>> type(w) == Vector
True
>>> print(w)
[[ 60 15 20 ]]
```

   Here, the left operand, 5, is an integer, so Python calls the "`__mul__()`" method of the `int` class. But that method does not know how to scale our custom Vector class, i.e. it does not know what to do when it receives a Vector instance as the other operand, and returns an object called `NotImplemented` in that case. So then, Python checks if the operand on the right side knows how to handle this operation as a right operand, i.e. if the Vector class has an implementation of the "`__rmul__()`" method.

   So, for this, you will need to implement the "`__rmul__()`" method.

   Hint: that method can rely on the implementation of the "`__mul__()`" method, rather than reimplementing the same, or very similar, code.

7. The class should implement the "`__repr__()`" method. This is similar to the "`__str__()`" method, in that it returns a string representation of the Vector object. But whereas the "`__str__()`" method should present the object in a manner that is readable to the user, the main goal of the "`__repr__()`" method is to present the object in an unambiguous manner to a programmer, for debugging purposes. That is, it should state which class the object is an instance of, as well as the values that distinguish it from other instances of the class.

A common convention, at least for simple classes such as this one, is to have the "`__repr__()`" method return a parsable string representation of the object, i.e. the name of the class, followed by its parameters in parenthesis, just as when instantiating a new instance of the class. This allows Python to recreate the object from the representation when `repr()` is used in conjunction with `eval()`, for example.

That is what we will do here. Thus, it should work as follows:

```
>>> v = Vector(12, 3, 4)
>>> repr(v)
'Vector(12, 3, 4)'
>>> u = eval(repr(v))
>>> type(u) == Vector
True
>>> print(u)
[[ 12 3 4 ]]
>>> u == v
True
```

8. (a) Relative comparison between two Vectors should also work, by comparing the lengths of the Vectors:

```
>>> u = Vector(12, 3, 4)
>>> v = Vector(3, 4, 5)
>>> u > v
True
>>> v > u
False
```

   For this, you will need to implement the "`__gt__()`" method.

   Note that it is not conventional to compare vectors based on their lengths, this is one possible way to interpret such comparison, but there might be others.

   (b) You should also overload more comparison operators, such as the >= operator:

```
>>> u = Vector(12, 3, 4)
>>> v = Vector(3, 4, 12)
>>> w = Vector(3, 4, 5)
>>> u >= v
True
>>> v >= u
True
>>> w >= u
False
```

   For this, you need to implement the "`__ge__()`" method. Notice that u and v are not the same Vector, but the comparison returns `True` because they have the same length. So this comparison is not compatible with the equality comparison.

   (c) And the < operator:

```
>>> u = Vector(12, 3, 4)
>>> v = Vector(3, 4, 5)
>>> u < v
False
>>> v < u
True
```

   For this, you will need to implement the "`__lt__()`" method.

   (d) And the <= operator:

```
>>> u = Vector(12, 3, 4)
>>> v = Vector(3, 4, 12)
```

```
>>> w = Vector(3, 4, 5)
>>> u <= v
True
>>> v <= u
True
>>> u <= w
False
```

For this, you need to implement the "`__le__()`" method.

9. As if all that wasn't enough, the class should also enable calculation of the dot product of two Vectors.

   We could use the `*` operator for that as well, but there's no need to overcrowd that operator, and it would complicate the "`__mul__()`" method, which would then need to check the type of its argument before deciding what to do with it, since we're already using it for Vector scaling.

   So, here we'll just implement a custom method called "`dot()`":

   ```
   >>> u = Vector(12, 3, 4)
   >>> v = Vector(1, 2, -3)
   >>> u.dot(v)
   6
   ```

   Recall that the dot product of two vectors $a = [a_1, a_2, a_3]$ and $b = [b_1, b_2, b_3]$ is calculated as:

   $$a \cdot b = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

10. And finally, the class should also be able to calculate the cross product of two Vectors:

    ```
    >>> u = Vector(12, 3, 4)
    >>> v = Vector(1, 2, 3)
    >>> w = u.cross(v)
    >>> type(w) == Vector
    True
    >>> print(w)
    [[ 1 -32 21 ]]
    ```

    The cross product of two 3-dimensional vectors $a = [a_1, a_2, a_3]$ and $b = [b_1, b_2, b_3]$ is a new 3-dimensional vector, and can be calculated with the following formula: $a \times b = \begin{bmatrix} a_2 \cdot b_3 - a_3 \cdot b_2 \\ a_3 \cdot b_1 - a_1 \cdot b_3 \\ a_1 \cdot b_2 - a_2 \cdot b_1 \end{bmatrix}$.