Rectas_2

April 22, 2021

1 Reimplementando la clase Recta

A estas alturas deberíais tener ya las Rectas implementadas de algún modo, con una instrucción de creación
que toma dos de sus puntos como argumentos. Aquí vamos a llamar a la clase Recta_1 (renombra tu implementación
para encajar aquí lo que ya tienes). La descripción de la clase empieza por tanto así:

[1]:
```
class Recta_1(object):
    def __init__(self,p1,p2):
        pass
```

Pero podríais haber implementado la recta con cualquier otra representación: mediante su pendiente p, y el punto de corte c con el eje y (dejando las rectas verticales como caso particular); mediante su ecuación *universal* `ax+by=c` (eventualmente normalizada exigiendo p.e. que a sea siempre 0 o 1); mediante dos de sus puntos; etc, etc.

Se trata de que implementeis las Rectas con otra representación diferente, utilizando como función de creación Recta_2(pt,p) donde pt es la pendiente
(que será 'infinito' para las rectas verticales) y p será un punto de la recta, perteneciente al eje y de ser ello posible.
La nueva clase empezará por tanto como sigue, y contará con atributos diferentes a los de *Clase_1*.

[2]:
```
class Recta_2(object):
    def __init__(self,pt,p):
        pass
```

Tras tener las rectas representadas mediante dos clases distintas nos gustaría tener métodos que nos permitieran cambiar de una a otra. Por ejemplo:

[3]:
```
class Recta_1(object):
...
    def formato_recta_2(self):
        # Rellenar el hueco con los cálculos necesarios
        # para crear una nueva recta en el formato de repesentación Recta_2
        return Recta_2(p, c)
```

Pero tener las rectas representadas de dos formas diferentes aumenta la probabilidad de cometer fallos, al mezclarlas inadvertidamente de forma incorrecta. Especialmente si las funciones de inicialización en ambos casos tienen dos argumentos, como hemos propuesto aquí.

[4]:
```
p1 = Point(1.0, 2.0)
p2 = Point(2.0, 3.0)

pt = 8.0
p = Point(0.0, 3.0)

r1 = Recta_1(p1, p2)
r2 = Recta_2(p1, p)
```

```
␣
↪---------------------------------------------------------------------------

       NameError                                 Traceback (most recent call␣
↪last)

       <ipython-input-4-43a8f62423a6> in <module>
   ----> 1 p1 = Point(1.0, 2.0)
         2 p2 = Point(2.0, 3.0)
         3
         4 pt = 8.0
         5 p = Point(0.0, 3.0)


       NameError: name 'Point' is not defined
```

Es muy fácil despistarnos si no prestamos atención. Este código no se "quejará" al crear las variables `r1` y `r2`, pero fallará estrepitosamente en cuanto invoquemos métodos con `r2`, que fue inadvertidamente creada muy malamente, al utilizar el punto `p1` como su pendiente. Una forma de tratar de evitar estos problemas es incorporando excepciones.

Modificar entonces las funciones de creación para controlar los tipos de los argumentos. Además, si no habéis llamado del mismo modo a los demás métodos de las dos clases (lo que ciertamente habría sido lo adecuado si habéis programado siguiendo las directrices de la *POO*), tendríais que andaros con ojo al invocar a los respectivos métodos para que no se produzcan errores. £Cómo lo haríais?

[ ]:
```

```