12 Exercises

September 13, 2020

0.1 Exercise 12.1

Create a class to represent vectors of arbitrary length and which is initialised with a list of values, e.g.:

`x = MyVector([0, 2, 4])`

Equip the class with methods that:

1. Return the length of the vector (use name `size`)
2. Compute the norm of the vector $\sqrt{x \cdot x}$ (use name `norm`)
3. Compute the dot product of the vector with another vector (use name `dot`)

Test your implementation using two vectors of length 3. To help you get started, a skeleton of the class is provided below. Don't forget to use `self` where necessary.

0.1.1 Solution

[0]:
```
import math  # Use this to get the sqrt

class MyVector:
    """A vector object that can return its size and norm, and can compute the␣
 ↪dot product
    with another vector  """

    def __init__(self, x):
        self.x = x

    # Return length of vector
    def size(self):
        return len(self.x)

    # This allows access by index, e.g. y[2]
    def __getitem__(self, index):
        return self.x[index]

    # Return norm of vector
    def norm(self):
        return math.sqrt(sum(i*j for i, j in zip(self.x, self.x)))
```

```
    # Return dot product of vector with another vector
    def dot(self, other):
        return sum(i*j for i, j in zip(self.x, other.x))
```

[0]:
```
# Create two vectors
u = MyVector([1, 1, 2])
v = MyVector([2, 1, 1])

print(u.size())
print(u.norm())
print(u.dot(v))

assert u.size() == 3
assert round(u.norm() - 2.449489742783178) == 0.0
assert round(u.dot(v) - 5.0, 10) == 0.0
```

```
3
2.449489742783178
5
```

0.2 Exercise 12.2

1. Create a class for holding a student record entry. It should have the following attributes:
   - Surname
   - Forename
   - Birth year
   - Tripos year
   - College
   - CRSid (optional field)
2. Equip your class with the method '`age`' that returns the age of the student in whole years
3. Equip your class with the method '`__repr__`' such using `print` on a student record displays with the format

   `Surname: Bloggs, Forename: Andrea, College: Churchill`
4. Equip your class with the method `__lt__(self, other)` so that a list of record entries can be sorted by (surname, forename) using the Python built-in sort function.

   Create a list of entries and test the sorting. Make sure you have two entries with the same surname.

   Recall that the methods starting with `__`, e.g. `__lt__` and `__repr__`, should **not** be called directly. Python with map them to other operations, e.g. `__repr__` is called when using `print`, and `__lt__` is called when using `<`. These functions must have a return value.

Hint: To get the current year:

[0]:
```
import datetime
year = datetime.date.today().year
print(year)
```

2020

0.2.1 Solution

[0]:
```
import datetime
class StudentEntry:
    def __init__(self, surname, forename, birth_year, year, college,␣
↪crsid=None):
        self.surname = surname
        self.forename = forename
        self.birth_year = birth_year
        self.year = year
        self.college = college
        self.crsid = crsid

    def age(self):
        return datetime.date.today().year - self.birth_year

    def returnReverseName(self):
        '''return the name of the player as last, first'''
        return self.surname, self.forename

    def __eq__ (self, other):
        '''determine if this persons name is the same as the other␣
↪personsname'''
        return self.returnReverseName() == other.returnReverseName()

    def __lt__(self,other):
        '''determine if this persons name is less than the other persons name␣
↪alphabetically'''
        return self.returnReverseName() < other.returnReverseName()

    def __gt__ (self, other):
        '''determine if this persons name is greater than the other persons␣
↪name alphabetically'''
        return self.returnReverseName() > other.returnReverseName()

    def __repr__(self):
        'Surname: Bloggs, Forename: Andrea, College: Churchill'
        return str('Surname: {}, Forename: {}, College: {}'.format(self.
↪surname, self.forename, self.college))
```

[0]:
```
s0 = StudentEntry("Bloggs", "Andrea", 1996, 1, "Churchill", "ab1001")
s1 = StudentEntry("Reali", "John", 1997, 1, "Corpus Christi")
s2 = StudentEntry("Bacon", "Kevin", 1996, 1, "Newnham")
s3 = StudentEntry("Bacon", "Alexander", 1996, 1, "Queens")
assert s0 < s1
assert s0 > s2
assert s3 < s2
assert s0.age() ==  datetime.date.today().year - 1996
assert s1.age() ==  datetime.date.today().year - 1997
assert str(s1) == "Surname: Reali, Forename: John, College: Corpus Christi"
```

[0]:
```
# Test sorting
s = [s0, s1, s2, s3]
s.sort()
for earlier, later in zip(s, s[1:]):
    assert earlier.surname <= later.surname
    if earlier.surname == later.surname:
        assert earlier.forename <= later.forename
print(s)
```

```
[Surname: Bacon, Forename: Alexander, College: Queens, Surname: Bacon, Forename:
Kevin, College: Newnham, Surname: Bloggs, Forename: Andrea, College: Churchill,
Surname: Reali, Forename: John, College: Corpus Christi]
```

0.3 Reference

https://stackoverflow.com/questions/36960379/alphabetizing-a-list-through-class-methods