Consider the mathematical structure–the vector. To maximize code reusability, you will implement the BetterVector class. The BetterVector class will be a subclass of the Vector class, which can be found here. The Vector Class already provides for addition which BetterVector will inherit, but you will need to provide for vector subtraction and multiplication (inner product). Use operator overloading. Implement the BetterVector class as noted above and test your newly created multiplication and subtraction operations in a "main method", if **name** == **main**

```
import collections

class Vector:
  """Represent a vector in a multidimensional space."""

  def __init__(self, d):
    if isinstance(d, int):
      self._coords = [0] * d
    else:
      try:                                     # we test if param is iterable
        self._coords = [val for val in d]
      except TypeError:
        raise TypeError('invalid parameter type')

  def __len__(self):
    """Return the dimension of the vector."""
    return len(self._coords)

  def __getitem__(self, j):
    """Return jth coordinate of vector."""
    return self._coords[j]

  def __setitem__(self, j, val):
    """Set jth coordinate of vector to given value."""
    self._coords[j] = val

  def __add__(self, other):
    """Return sum of two vectors."""
    if len(self) != len(other):          # relies on __len__ method
      raise ValueError('dimensions must agree')
    result = Vector(len(self))           # start with vector of zeros
    for j in range(len(self)):
      result[j] = self[j] + other[j]
    return result
  def __eq__(self, other):
    """Return True if vector has same coordinates as other."""
    return self._coords == other._coords

  def __ne__(self, other):
    """Return True if vector differs from other."""
    return not self == other             # rely on existing __eq__ definition

  def __str__(self):
    """Produce string representation of vector."""
    return '<' + str(self._coords)[1:-1] + '>'  # adapt list representation

  def __neg__(self):
    """Return copy of vector with all coordinates negated."""
    result = Vector(len(self))           # start with vector of zeros
    for j in range(len(self)):
      result[j] = -self[j]
    return result

  def __lt__(self, other):
    """Compare vectors based on lexicographical order."""
    if len(self) != len(other):
      raise ValueError('dimensions must agree')
    return self._coords < other._coords

  def __le__(self, other):
    """Compare vectors based on lexicographical order."""
    if len(self) != len(other):
      raise ValueError('dimensions must agree')
    return self._coords <= other._coords

if __name__ == '__main__':
  # the following demonstrates usage of a few methods
  v = Vector(5)              # construct five-dimensional <0, 0, 0, 0, 0>
  v[1] = 23                  # <0, 23, 0, 0, 0> (based on use of __setitem__)
  v[-1] = 45                 # <0, 23, 0, 0, 45> (also via __setitem__)
  print(v[4])                # print 45 (via __getitem__)
  u = v + v                  # <0, 46, 0, 0, 90> (via __add__)
  print(u)                   # print <0, 46, 0, 0, 90>
  total = 0
  for entry in v:            # implicit iteration via __len__ and __getitem__
```

```
        total += entry
```

```
45
<0, 46, 0, 0, 90>
```

```
class BetterVector(Vector):
    """Will need to provide for vector subtraction and multiplication (inner product). Use
    def __init__(self, d): # d is the dimension of the vector
        super().__init__(d) # call the parent class's __init__ method

    def __sub__(self, other): # other is the vector to be subtracted from self
        """Return difference of two vectors."""
        if len(self) != len(other): # it is against the rules to add vectors of different
            raise ValueError('dimensions must agree') # raise an exception
        result = Vector(len(self)) # start with vector of zeros
        for j in range(len(self)): # iterate over the coordinates
            result[j] = self[j] - other[j] # subtract the coordinates
        return result # return the result

    def __mul__(self, other): # other is the vector to be multiplied by self
        """Return inner product of two vectors."""
        if len(self) != len(other): # it is against the rules to add vectors of different
            raise ValueError('dimensions must agree') # raise an exception
        result = Vector(len(self))  # start with a zero for result (to initialize the vari
        for j in range(len(self)): # iterate over the coordinates
            result[j] = self[j] * other[j] # add the product of the coordinates
        return result # return the result
```

Implement the BetterVector class as noted above and test your newly created multiplication and subtraction operations in a "main method", if **name** == **main**

```
v = BetterVector(5)
v[0] = 0
v[1] = 1
v[2] = 2
v[3] = 3
v[4] = 4

print(v*v)
```

```
<0, 1, 4, 9, 16>
```

```
if __name__ == '__main__':
  v = BetterVector(5)
  v[0] = 0
  v[1] = 1
  v[2] = 2
  v[3] = 3
  v[4] = 4
  print(v[4])
  u = v - v
  print(u)
  x = v * v
  print(x)
  total = 0
  for entry in v:
    total += entry
```

```
4
<0, 0, 0, 0, 0>
<0, 1, 4, 9, 16>
```