Arrangement of boxes

In this lab, you will implement a class called `Truck` containing a collection of `Box` objects. Any `Box` object has attributes `length`, `width`, and `height`. The class `Truck` should be able to sort contained boxes based on given criteria and rotate the sequence either to the right or to the left.

Box class

We build a `Box` class with the following ADT.

- `__init__(self, length, width, height)` - initialize a `Box` object with `length`, `width`, and `height` attributes.
- `volume(self)` - return the volume of the `Box`
- `__lt__(self, other)` - return `True` if the volume of `self` is less than the volume of `other`
- `__eq__(self, other)` - return `True` if and only if `self` and `other` have the same volume.
- `__le__(self, other)` - return `True` if the volume of `self` is not greater than the volume of `other`.

Start by writing a class called `Box` and store it in a file called `boxarrangement.py`. Then, implement its methods using the above ADT.

Truck Class.

It stores a collection of boxes in a sequence. We create a `Truck` class with the following ADT.

- `__init__(self, boxes)` - initialize a new truck with the list `boxes`.
- `sort(self, key=None)` - Sort the containing boxes in-place based on a given key function.
- `sortbyvolume(self)` - Sort the boxes in-place based on their volume.
- `sortbylength(self)` - Sort the boxes in-place based on their length.
- `sortbyheight(self)` - Sort the boxes in-place based on their height.
- `rotate(self, k)` - Rotate the sequence of boxes `k` steps.

Add a class called `Truck` to your `boxarrangement.py` file and implement `__init__` method. Note that `Truck` **does not extend** `Box` class and it only **contains** a list of boxes

List sort in python

In python, `list` built-in type has a method called `sort`. By default, `sort` method uses the `__lt__` comparator of the objects in the list to sort the list. This method returns nothing and sorts the list in-place.

```
L = [8, 2, 5, 7, 6, 1, 3]
L.sort()
print(L)
```

```
[1, 2, 3, 5, 6, 7, 8]
```

Furthermore, `sort` receives two keyword arguments called `key` and `reverse`. If we set `reverse = True`, then the list will be sorted in the reverse order. The `key` argument accepts a function that will be called on each list element before making comparisions.

```
L = [(1,2), (4,5), (0,100), (2, 100)]

def prod(p):
    return p[0] * p[1]

L.sort(key = prod)
print(L)
```

```
[(0, 100), (1, 2), (4, 5), (2, 100)]
```

Python also supports anonymous in-line functions called `lambda`. For example, you may write a lambda expression `lambda p: p[0] * p[1]` which is equivalent to the `prod` function. So, we can write the previous code succinctly using a lambda expression in the following way.

```
L = [(1, 2), (4, 5), (0, 100), (2, 100)]
L.sort(key = lambda p: p[0] * p[1])
print(L)
```

```
[(0, 100), (1, 2), (4, 5), (2, 100)]
```

Now, implement `sort`, `sortbyvolume`, `sortbylength`, and `sortbyheight` for `Truck` class using `list.sort` method.

Rotate a sequence

The `rotate` method of `Truck` receives an integer `k` and rotates `boxes` as many as `k` steps. If `k` is greater than zero, then `rotate` should rotate the boxes `k` steps to the right. For `k` less than zero, we

rotate the boxes `k` steps to the left. Note that the rotation is circular. In other words, if an element exits from one side, then it will be added to the list from the other side. For example, let `L = [1, 2, 3, 4, 5, 6]`. Then, the rotated sequences for `k=3` and `k=-2` are `L = [4, 5, 6, 1, 2, 3]` and `L=[3, 4, 5, 6, 1, 2]`, respectively.

Implement the `rotate` function. This method runs in $O(n)$ time.

Challenge (Extra)

The `rotate` function can be done **in-place** in linear time. This means that we don't make a copy of the list, and instead, we modify the list by moving elements around. It will require using some ideas from Euclid's greatest common divisor algorithm. Can you do it?

Summary

For this lab you should implement `Box` and `Truck` classes with the provided ADT and store them in `boxarrangement.py`. Then you need to submit `boxarrangement.py` to Mimir.