1. Perkovic 8.19, 1st ed.

8.19 Implement your own string class \texttt{myStr} that behaves like the regular \texttt{str} class except that:

- The addition (+) operator returns the sum of the lengths of the two strings (instead of the concatenation).
- The multiplication (*) operator returns the product of the lengths of the two strings.

The two operands, for both operators, are assumed to be strings; the behavior of your implementation can be undefined if the second operand is not a string.

```python
>>> x = myStr('hello')
>>> x + 'universe'
13
>>> x * 'universe'
40
```

8.20 Develop a class myList that is a subclass of the built-in list class. The only difference between myList and list is that the sort method is overridden. myList containers should behave just like regular lists, except as shown next:

```python
>>> x = myList([1, 2, 3])
>>> x
[1, 2, 3]
>>> x.reverse()
>>> x
[3, 2, 1]
>>> x[2]
1
>>> x.sort()
You wish...
```
3. Perkovic 8.25

8.25 Implement class Worker that supports methods:

- `__init__()` : Constructor that takes as input the worker's name (as a string) and the hourly pay rate (as a number)
- `changeRate()` : Takes the new pay rate as input and changes the worker's pay rate to the new hourly rate
- `pay()` : Takes the number of hours worked as input and prints 'Not Implemented'

Next develop classes HourlyWorker and SalariedWorker as subclasses of Worker. Each overloads the inherited method `pay()` to compute the weekly pay for the worker. Hourly workers are paid the hourly rate for the actual hours worked; any overtime hours above 40 are paid double. Salaried workers are paid for 40 hours regardless of the number of hours worked. Because the number of hours is not relevant, the method `pay()` for salaried workers should also be callable without an input argument.

```python
>>> w1 = Worker('Joe', 15)
>>> w1.pay(35)
Not implemented
>>> w2 = SalariedWorker('Sue', 14.50)
>>> w2.pay()
580.0
>>> w2.pay(60)
580.0
>>> w3 = HourlyWorker('Dana', 20)
>>> w3.pay(25)
500
>>> w3.changeRate(35)
>>> w3.pay(25)
875
```
4. Perkovic 8.26

8.26 Create a class `Segment` that represents a line segment in the plane and supports methods:

- `__init__`: Constructor that takes as input a pair of `Point` objects that represent the endpoints of the line segment
- `length`: Returns the length of the segment
- `slope`: Returns the slope of the segment or `None` if the slope is unbounded

```python
>>> p1 = Point(3, 4)
>>> p2 = Point()
>>> s = Segment(p1, p2)
>>> s.length()
5.0
>>> s.slope()
0.75
```