ISTA 350 Magic Methods Worksheet Name:

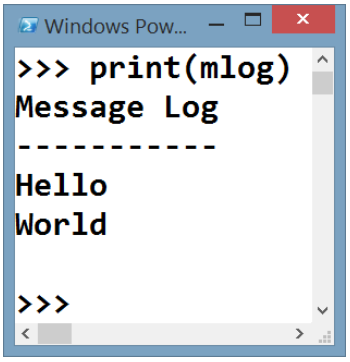
Define a class called `MessageLog`. It contains four instance methods:

`init`: Initialize a new `MessageLog` instance. A `MessageLog` object contains one instance variable, a list of strings. `init` takes an argument that is a list of strings with a default value of `None`. If the list argument is empty or `None`, initialize the instance variable to an empty list. Otherwise, set it to a copy of the list argument.

`add_message`: takes a string argument and appends it the log.

`__add__`: exactly the same as `add_message` except for the name.

`repr`: return a string representation of the instance that prints like this:



Define a class called `Polygon`. Use the `total_ordering` decorator.

`init`: Initialize a new `Polygon` instance. A `Polygon` object contains two instance variables: the polygon's number of sides and its area. `init` has corresponding parameters with default values of `None`. If either argument is `None`, get values from the user.

`from_sidelen`: This class method takes a number of sides and a side length and returns a new `Polygon`. You will need to calculate the area in order to make the new instance (use math.pi). If `n` is the number of sides and `s` is the side length, the formula is

$$Area = \frac{s^2 n}{4\tan(\frac{\pi}{n})}$$

`add`: Takes a `polygon` as an argument and returns a new `polygon` whose area is the sum of `self`'s and the argument's and whose number of sides is also the sum of the sides of the addends.

`eq`: Two `polygon`s are equal if their number of sides times their areas are equal.

`lt`: One polygon is less than another if its number of sides times its area is less than that of the other.

Define a class called `Person`. It contains four instance methods:

`init`: Each `Person` object has four instance variables, called `first`, `last`, `bday`, and `email`, and `init` has four corresponding string parameters, each of which has the empty string as a default argument. The first parameter is the `Person`'s first name, the second the last name, the third his/her birthday, and the last the `Person`'s e-mail. If any of the parameters are the empty string, get a value from the user using one of the following prompts:

```
Enter person's first name:
Enter person's last name:
Enter person's birthday:
Enter person's e-mail:
```

Each colon is followed by a space.

`repr`: This method returns a string in the following format: `'Rich Thompson: 5/21, rm@g'`.

`read_person`: This is a class method that reads the data necessary from a text file to create and return a `Person` instance. It takes one argument, a file object (not a filename). It reads a line from the file. If the line is empty, return `False`. Otherwise, use the contents of this and the next three lines as the first name, last name, birthday, and e-mail of a new `Person`. Remember to use the `classmethod` decorator.

This method allows one to traverse a file of `Person` objects, or perhaps `Person` objects with some other data interspersed, and read and create one `Person` object from it at a time.

`write_person`: This instance method takes one argument, a file object. It writes the instance variables, one per line, to the file in this order: first, last, birthday, email.

Define a class called `Counter` that represents an event counter. It needs functionality that allows the user to add to the number of events and functionality to return the count of the events that have occurred in the last minute. Therefore, we will keep a list of `datetime` objects so that we can both count them and know when they occurred. Call this list `self.events`. It is the only instance variable. The initializer takes an event list with the default argument of the empty list. Assign this parameter to `self.events`. The unary positive operator (the plus sign with a single operand to its right, e.g. `+4`) magic method, `pos`, appends the current time to the end of the list. `datetime.now()` returns the current time. The augmented assignment operator (`+=`) magic method, `iadd`, is a bit tricky. The operand will be an integer. Append the current time the number of times represented by the operand. The `get_count` instance method takes no arguments. Get the time a minute ago by subtracting `timedelta(minutes=1)` from the current time. Remove all `datetime` objects older than a minute ago from the events list. Return the length of the list.

```
from datetime import datetime, timedelta
```