object A compound data type that is often used to model a thing or concept in the real world. It bundles together the data and the operations that are relevant for that kind of data. Instance and object are used interchangeably.

object-oriented programming A powerful style of programming in which data and the operations that manipulate it are organized into objects.

object-oriented language A language that provides features, such as user-defined classes and inheritance, that facilitate object-oriented programming.

15.12 Exercises

1. Rewrite the `distance` function from the chapter titled *Fruitful functions* so that it takes two `Points` as parameters instead of four numbers.
2. Add a method `reflect_x` to `Point` which returns a new `Point`, one which is the reflection of the point about the x-axis. For example, `Point(3, 5).reflect_x()` is (3, -5)
3. Add a method `slope_from_origin` which returns the slope of the line joining the origin to the point. For example,

   ```
   >>> Point(4, 10).slope_from_origin()
   2.5
   ```

   What cases will cause this method to fail?
4. The equation of a straight line is "y = ax + b", (or perhaps "y = mx + c"). The coefficients a and b completely describe the line. Write a method in the `Point` class so that if a point instance is given another point, it will compute the equation of the straight line joining the two points. It must return the two coefficients as a tuple of two values. For example,

   ```
   >>> print(Point(4, 11).get_line_to(Point(6, 15)))
   >>> (2, 3)
   ```

   This tells us that the equation of the line joining the two points is "y = 2x + 3". When will this method fail?
5. Given four points that fall on the circumference of a circle, find the midpoint of the circle. When will this function fail?

   Hint: You *must* know how to solve the geometry problem *before* you think of going anywhere near programming. You cannot program a solution to a problem if you don't understand what you want the computer to do!
6. Create a new class, SMS_store. The class will instantiate SMS_store objects, similar to an inbox or outbox on a cellphone:

   ```
   my_inbox = SMS_store()
   ```

   This store can hold multiple SMS messages (i.e. its internal state will just be a list of messages). Each message will be represented as a tuple:

```
(has_been_viewed, from_number, time_arrived, text_of_SMS)
```

The inbox object should provide these methods:

```
my_inbox.add_new_arrival(from_number, time_arrived, text_of_SMS)
  # Makes new SMS tuple, inserts it after other messages
  # in the store. When creating this message, its
  # has_been_viewed status is set False.

my_inbox.message_count()
  # Returns the number of sms messages in my_inbox

my_inbox.get_unread_indexes()
  # Returns list of indexes of all not-yet-viewed SMS messages

my_inbox.get_message(i)
  # Return (from_number, time_arrived, text_of_sms) for message[i]
  # Also change its state to "has been viewed".
  # If there is no message at position i, return None

my_inbox.delete(i)      # Delete the message at index i
my_inbox.clear()        # Delete all messages from inbox
```

Write the class, create a message store object, write tests for these methods, and implement the methods.