Exercise 1.5: Object-Oriented Programming in Python

Learning Goals

- Apply object-oriented programming concepts to your Recipe app

Reflection Questions

1. In your own words, what is object-oriented programming? What are the benefits of OOP?

   In OOP, the code is organized in classes which represent objects. It follows the DRY-principle to keep your code clean, readable and maintainable.

2. What are objects and classes in Python? Come up with a real-world example to illustrate how objects and classes work.
   Everything in Python is an object. Objects can contain data attributes (variables) and procedural attributes (methods = functions).

   Let's take the recipe script, the task for exercise 1.5:

   A cooking recipe is represented through a "Recipe"-class.
   It has a class variable (a variable that can be accessed by each instance of the class) :

```
all_ingredients = []
```

   It also has four normal data attributes:

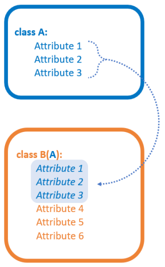
   They are unique to each instance and represent properties of the object (the recipe).

```
self.name = name
self.cooking_time = 0
self.ingredients = []
self.difficulty = ""
```

   Procedural attributes (methods):
   We have getter and setter methods to set or read the value of the variables they are written for. (set_name and get_name for example).

```
def calc_difficulty(self):
def set_name(self, name):
def set_cooking_time(self, cooking_time):
def get_name(self):
def get_cooking_time(self):
def get_ingredients(self):
def get_difficulty(self):
def add_ingredients(self, *ingredients):
def search_ingredient(self, ingredient):
def update_all_ingredients(self):
    def __str__(self): // returns a string so the object can be printed
```

3. In your own words, write brief explanations of the following OOP concepts; 100 to 200 words per method is fine.

Method	Description
Inheritance	If a class is created via "`class Class_Name(Parent_Class_Name)`" it inherits all attributes from it's parent class. Those attributes can be overwritten, but don't have to be. Also the class can be expanded by adding new ones.  class A: Attribute 1 Attribute 2 Attribute 3 class B(A): Attribute 1 Attribute 2 Attribute 3 Attribute 4 Attribute 5 Attribute 6
Polymorphism	Different classes can have procedural attributes (methods) with the same name. Although they have the same name, they can perform different operations.

Operator Overloading	As classes are more complex that just a string or just a number, we have to define what the "+" operator (or any other operator) should do exactly if used on instances of a class.
	
	E.g.:
	
	`class Person():`
	`    def __init__(self, name, height, weight)`
	`        self.name = name`
	`        self.height = height`
	`        self.weight = weight`
	
	`    def __add__(self, other):`
	`        name = self.name + other.name`
	`        height = self.height + other.height`
	`        weight = self.weight + other.weights`
	`        return Person(name, height, weight)`
	
	`personA = Person("A", 180,  80)`
	`personB = Person("B", 150, 50)`
	
	`#  If we did not overload the + operator, this would not work:`
	`personC = personA + personB`