Tasks:

1. (Easy) Implement an object that sums elements of an **int** (or **float**) **array**. Use the **reduce** attribute object of the **array** object (i.e., **array.reduce**).
2. (Middle) Implement an object that finds the product of integers numbers in an **array**. The object must be **<u>recursive</u>**.
3. (Middle+) Implement an object that finds conjunction of **booleans** in an **array** using **array.reduce**. The object must short-circuit. This means that the **reduce** object must not compute any sub-conjunctions when the result is obvious (=0). ***<u>Hint</u>***: Instead, the **reduce** object must just pass the answer until the array is processed with no sub-computations.
4. (Middle++) Implement an object that finds disjunction of **booleans** in an **array**. The object must be **<u>tail-recursive</u>**. Also, the object must short-circuit. This means that the object must not compute any sub-disjunctions when the result is obvious (=1). ***<u>Hint</u>***: Instead, the object must just "return" the answer at that point.
5. (Middle) Implement an object that concatenates **strings** in an **array**. Use **array.reduce**. ***<u>Hint:</u>*** use **sprintf**.
6. (Middle) Given an **array** of **float** objects representing temperature values in Celsius. Implement an object that finds the corresponding **array** of temperature values in Fahrenheit. Use **array.map**. ***<u>Hint:</u>*** the formula is (x**°C** × 9.0/5.0) + 32.0 = y**°F**
7. (Middle+) Given an **array** of **ints**. Implement an object that finds the first (in other words, the leftmost) minimum in the array. The object must return another object with two attributes: index and value. Use **array.reduce**.
8. (Middle++) Given an **array** of **ints** and the **minimum** object from Task 7. Implement the **removeAt** object that removes the element at position **index** and returns the resulting array. Use **array.reducei** (make sure that you are using **reducei**, not **reduce**).
9. (Hard) Given an **array** of **ints** and objects **minimum** and **removeAt** from Tasks 7, 8. Implement the selection sorting algorithm.
10. (Easy) Given an **array** of **objects**. Implement the **reverse** object that reverses the order of objects from a, b, c, d, …, x, y, z to z, y, x, …, c, b, a. The object must be recursive.
11. (Hard) Make your own **.reduce** operation for arrays from scratch. Implement an object that performs the **reduce** operation over an array (contents of the array may be of any type). Use recursion or tail-recursion.
12. (Hard) Make your own **.map** operation for arrays from scratch. Implement an object that performs the **map** operation over an array (contents of the array may be of any type). Use recursion or tail-recursion.
13. (Hard+) Make your own **.map** operation for arrays from scratch. Implement an object that performs the **map** operation over an array (contents of the array may be of any type). Use your own **.reduce** operation from Task 12 or the standard **array.reduce** (the choice is up to you and does not matter). Do not use recursion of any kind! Use only the **.reduce** operation to implement **.map**! Hint: **.reduce** transforms arrays (N) to a sole value (1), that's why it is referred to as N-to-1 transformation operation. But nothing stops us to consider the sole output value (1) as another array!

Tests:

1. What elemental operation of the EO language may be used for object creation?
    a. **Abstraction.** // **Yes!** Abstraction may be compared with class declaration in Java. However, EO has no classes. The object defined through abstraction is a real object that may be used. So, yes. Abstraction creates objects (and allows us to define the structure of the object).
    b. **Decoration.** // **No :(** Decoration is used to compose objects. This means that the object A decorated by the object B inherits (or extends) B's attributes and adds its own attributes (that may possibly hide/shadow the original attributes of B).
    c. **Dataization.** // **No!** Even though the actual dataization algorithm **may** instantiate objects in the target environment (for instance, inside JVM), these are implementation details of a concrete EO compiler. The EO language is **declarative**. We can't manage the actual process of execution of our programs. The operation of dataization is the only one we can't control. It's done by the EO environment based on some rules.
    d. **Application.** **Yes!** We can create copies of objects through application. Also, we can bind values with free attributes thanks to this operation. However, we can't define the structure of objects when we apply them. That's the main and principal difference when comparing this operation to abstraction.
2. What array attributes are **normally** used to implement N-to-N transformations?
    a. array.reduce.
    b. array.length.
    c. array.empty.
    d. **array.map**.
    e. array.reducei.
    f. **array.mapi**.
3. What array attributes are **normally** used to implement N-to-1 transformations?
    a. **array.reduce.**
    b. array.length.
    c. array.empty.
    d. array.map.
    e. **array.reducei.**
    f. array.mapi.
4. (Tricky question). What array attributes **may be possibly** used to implement N-to-N transformations?
    a. **array.reduce.**
    b. array.length.
    c. array.empty.
    d. **array.map**.
    e. **array.reducei.**
    f. **array.mapi**.
5. (Tricky question). What array attributes **may be possibly** used to implement N-to-1 transformations?

a. **array.reduce.**
b. array.length.
c. array.empty.
d. array.map.
e. **array.reducei.**
f. array.mapi.