My Report

Learn about collections in Pharo and their iterators

A collection groups objects. It is used to store and organize data.

Pharo has arrays, ordered collections, sets, bags, dictionaries.

You iterate with :
- do: executes a block for each element.
- collect: transforms elements and returns a new collection.
- select: filters elements that match a condition.
- detect: finds the first element matching a condition.
- inject:into: folds/reduces a collection into one value.

I found this by looking up documentation and explanations online (like on stack overflow).

Learn about conditionals in Pharo

Pharo does not have if statements like Java. Instead, true and false are objects that understand messages like ifTrue: and ifFalse:.

```
(x > 5) ifTrue: [ 'big' ] ifFalse: [ 'small' ].
```

The difference from other programming languages, java for example, is that in java, the if statement is a keyword, whereas here, it's an object.

Benefit: any object could implement its own conditional behavior.
Drawback: it's very confusing at first, especially for other OOP languages developers like java devs.

Learn how to create classes and methods

Classes are defined in packages and methods on instance or class side.

```
Object << #Counter
	slots: { #count };
	package: 'MyCounter'.

increment   count := count + 1.

decrement   count := count - 1.
```

I did the exercises where I had to write a Counter class to learn about those.

My main issues were the class side vs instance side, which was confusing at first, and I struggled to set up my github repository. But thanks to Guiche, I managed to make it work, here is the link to access the repository :
https://github.com/ETS-Ronan/MyCounter-C3P

Learn about the basic Pharo coding style.

Common rules to follow :
- Small, readable methods
- Clear names
- Protocols group methods
- Not properly formatted

Code Critics, which is integrated in Pharo, shows violations.

Bad method example :
doItAll
count := 0. count := count + 1. count := count + 1. count := count - 1. self printOn: (String new writeStream). ^ count

Extras

Cascade : sending multiple messages without rewriting the variable name everytime

```
c := Counter new
        count: 5;
        increment;
        decrement.
```

Block closure : if I understood it correctly, it's kinda like a maths function, and it can also be passed to a collection, so the collection can apply it to each of its elements.

```
square := [ :x | x * x ].

#(1 2 3 4) collect: [ :n | n * 2 ].
"Result: #(2 4 6 8)"
```

I found this by looking up documentation and explanations online once again. Unfortunately I couldn't ask questions on the discord since I remembered last second to do this assignment…