



d08

## Object-Oriented Programming 2

42 staff [pedago@staff.42.fr](mailto:pedago@staff.42.fr)  
nate [alafouas@student.42.fr](mailto:alafouas@student.42.fr)

*Abstract: This is the subject for d08 of the OCaml pool.*

# Contents

<b>I</b>	<b>Ocaml piscine, general rules</b>	<b>2</b>
<b>II</b>	<b>Day-specific rules</b>	<b>4</b>
<b>III</b>	<b>Foreword</b>	<b>5</b>
<b>IV</b>	<b>Exercise 00: Atoms</b>	<b>6</b>
<b>V</b>	<b>Exercise 01: Molecules</b>	<b>8</b>
<b>VI</b>	<b>Exercise 02: Alkanes</b>	<b>10</b>
<b>VII</b>	<b>Exercise 03: Reactions</b>	<b>11</b>
<b>VIII</b>	<b>Exercise 04: Alkane combustion</b>	<b>12</b>
<b>IX</b>	<b>Exercise 05: Incomplete combustion</b>	<b>14</b>

# Chapter I

## Ocaml piscine, general rules

- Every output goes to the standard output, and will be ended by a newline, unless specified otherwise.
- The imposed filenames must be followed to the letter, as well as class names, function names and method names, etc.
- Unless otherwise explicitly stated, the keywords `open`, `for` and `while` are forbidden. Their use will be flagged as cheating, no questions asked.
- Turn-in directories are `ex00/`, `ex01/`, ..., `exn/`.
- You must read the examples thoroughly. They can contain requirements that are not obvious in the exercise's description.
- Since you are allowed to use the `OCaml` syntaxes you learned about since the beginning of the piscine, you are not allowed to use any additional syntaxes, modules and libraries unless explicitly stated otherwise.
- The exercises must be done in order. The graduation will stop at the first failed exercise. Yes, the old school way.
- Read each exercise FULLY before starting it ! Really, do it.
- The compiler to use is `ocaml-opt`. When you are required to turn in a function, you must also include anything necessary to compile a full executable. That executable should display some tests that prove that you've done the exercise right.
- Remember that the special token `;;` is only used to end an expression in the interpreter. Thus, it must never appear in any file you turn in. Anyway, the interpreter is a powerful ally, learn to use it at its best as soon as possible !
- The subject can be modified up to 4h before the final turn-in time.
- In case you're wondering, no coding style is enforced during the `OCaml` piscine. You can use any style you like, no restrictions. But remember that a code your peer-

evaluator can't read is a code she or he can't grade. As usual, big fonctions is a weak style.

- You will NOT be graded by a program, unless explicitly stated in the subject. Therefore, you are afforded a certain amount of freedom in how you choose to do the exercises. Anyway, some piscine day might explicitly cancel this rule, and you will have to respect directions and outputs perfectly.
- Only the requested files must be turned in and thus present on the repository during the peer-evaluation.
- Even if the subject of an exercise is short, it's worth spending some time on it to be absolutely sure you understand what's expected of you, and that you did it in the best possible way.
- By Odin, by Thor ! Use your brain !!!

# Chapter II

## Day-specific rules

- For every exercise you are required to turn-in a full program, with examples to prove your classes are working correctly. You may use any function or operator you want or see fit to use in your examples.
- Each exercise in today's subject is meant to be a sequel to the previous one; as a consequence, you will likely need to use your previous exercises to solve the one you're working on.
- As an obvious consequence, any exercise with 0 points for any reason (not turned in, does not compile, crashes, etc.) means the defence will not continue any further.
- You are generally free to use any type of data structure you want. But every time a method returns an associative list, it must be sorted in ascending order by its index.
- All chemical formulae for molecules must be written using the Hill notation.
- All the classes you implement today must be functional. Any imperative class in your code means no points for the entire exercise.
- Your code will not be modified during defences, which means a non-tested feature is a non-functional feature.

# Chapter III

## Foreword

It was a pleasure to burn.

It was a special pleasure to see things eaten, to see things blackened and changed. With the brass nozzle in his fists, with this great python spitting its venomous kerosene upon the world, the blood pounded in his head, and his hands were the hands of some amazing conductor playing all the symphonies of blazing and burning to bring down the tatters and charcoal ruins of history. With his symbolic helmet numbered 451 on his stolid head, and his eyes all orange flame with the thought of what came next, he flicked the igniter and the house jumped up in a gorging fire that burned the evening sky red and yellow and black. He strode in a swarm of fireflies. He wanted above all, like the old joke, to shove a marshmallow on a stick in the furnace, while the flapping pigeon-winged books died on the porch and lawn of the house. While the books went up in sparkling whirls and blew away on a wind turned dark with burning.


Montag grinned the fierce grin of all men singed and driven back by flame.

He knew that when he returned to the firehouse, he might wink at himself, a minstrel man, burnt-corked, in the mirror. Later, going to sleep, he would feel the fiery smile still gripped by his face muscles, in the dark. It never went away, that smile, it never ever went away, as long as he remembered.

By Ray Bradbury, in *Fahrenheit 451*.

# Chapter IV

## Exercise 00: Atoms

	Exercise 00
Our whole universe was in a hot, dense state...	
Turn-in directory : <i>ex00/</i>	
Files to turn in : *.ml, Makefile	
Forbidden functions : None	
Remarks : n/a	

You will write a virtual class named `atom` for...an atom. Today we're doing some chemistry! An atom is defined by a few things:

- A `name`, of type `string`.
- A `symbol`, of type `string`.
- An `atomic_number`, of type `int`.

All these things have to be methods, and you have to be able to set what they return through the atom's constructor. Of course, feel free to look up what an atom is if you don't understand what these are.

Your class will contain at least a `to_string` method to describe shortly what your atom is, and an `equals` method to compare atoms; aside from that, you can implement anything you think is suitable or you may need in the next exercises.

To go along with your atom virtual class, you will also write some real atoms, which will of course inherit your `atom` class. You have to write at least the following classes:

- `hydrogen`
- `carbon`
- `oxygen`
- And three more atoms of your choice, as long as they really exist.




Bonus points if you write more atoms. Learning important atoms is good for you.



# Chapter V

## Exercise 01: Molecules

	Exercise 01
Then nearly fourteen billion years ago expansion started. Wait...	
Turn-in directory : <i>ex01/</i>	
Files to turn in : *.ml, Makefile	
Forbidden functions : None	
Remarks : n/a	

Now that you have your atoms, you can make some molecules! If you don't know what a molecule is, please look it up before beginning this exercise.

Your molecule class is virtual, and it needs at least the following things:

- A name, of type `string`.
- A formula, of type `string`.

All these things have to be methods. Of course, feel free to look up what a molecule is if you don't understand what these are.

One very important constraint: you have to store the molecule's atoms internally and this atom storage must be used to compute the molecule's chemical formula. Also, the chemical formula must be formatted using the **Hill notation**. As such, your constructor will only accept a name and a list of atoms.

The chemical formula is simple: it's a small string which describes what atoms and how many of them are contained in the molecule. Let's build an example with **Trinitrotoluene** (or TNT for short). We get the list of atoms, and we end up with:

- 3 atoms of Nitrogen
- 5 atoms of Hydrogen
- 6 atoms of Oxygen
- 7 atoms of Carbon

We know the symbols of these atoms are N, H, O and C. All we have to do is enumerate their symbols and their quantity, right? So you would get something like:  $\text{N}_3\text{H}_5\text{O}_6\text{C}_7$ . But **NO! WAIT!** That's not how it works. The Hill notation says we get carbon, then hydrogen, then everything else in alphabetical order. So the result is:  $\text{C}_7\text{H}_5\text{N}_3\text{O}_6$ . Of course, you can't really write subscripts in a terminal, so writing it behind the symbol as it is is fine, like so: `C7H5N3O6`.

Your class will contain at least a `to_string` method to describe shortly what your molecule is, and an `equals` method to compare molecules; aside from that, you can implement anything you think is suitable or you may need in the next exercises.

To go along with your molecule class, you will write some real molecules, which will of course inherit your `molecule` class. You have to write at least the following classes:


- Water ( $\text{H}_2\text{O}$ )
- Carbon dioxide ( $\text{CO}_2$ )
- And three more molecules of your choice, as long as they really exist.



Bonus points if you write complex molecules. You don't want to be lazy, do you?

# Chapter VI

## Exercise 02: Alkanes

	Exercise 02
The Earth began to cool, the autotrophs began to drool...	
Turn-in directory : <i>ex02/</i>	
Files to turn in : *.ml, Makefile	
Forbidden functions : None	
Remarks : n/a	

Molecules are cool, but today we're focusing on a particular type of molecules, which are alkanes. Alkanes are a family of simple molecules composed of just carbon and hydrogen, which means we can create alkanes easily! The formula of an acyclic alkane is  $C_nH_{2n+2}$ . The name of the alkane simply depends on the value you give to  $n$ .

As such, your alkane's constructor only needs one parameter, which is  $n$ . It must be able to guess the name and the formula from this only  $n$  parameter.

Note that an alkane is still a molecule, which means you still have to provide the following methods:


- name
- formula
- to\_string
- equals

To go along with your alkane class, you will write...some real alkanes! As usual. Of course, they will inherit your **alkane** class and you will write at least the following ones:

- methane
- ethane
- octane

# Chapter VII

## Exercise 03: Reactions

	Exercise 03
Neanderthals developed tools, we built a wall (we built the pyramids!)	
Turn-in directory : <i>ex03/</i>	
Files to turn in : *.ml, Makefile	
Forbidden functions : None	
Remarks : n/a	

Now we're doing interesting things! Chemical reactions. A chemical reaction is a simple thing, really. You have some molecules at the start of your reaction, and you have...some molecules at the end. There is only one really important rule to follow, which is: you must have the same exact amount of atoms at the start and at the end of your reaction. "Nothing is lost, nothing is created, everything is transformed". If you've never heard that sentence, look up Antoine de Lavoisier and what he has done for chemistry.

Basically, you can instantiate a new chemical reaction with a list of molecules. Then you can call a `get_result` method to get the result.


All that explanation was good, but now let's talk about code! You will write a virtual class named `reaction`, which will be instantiated with two collections of molecules, one for the start and one for the end of your reaction.

Your class will also provide the following virtual methods:

- `get_start: (molecule * int) list`
- `get_result: (molecule * int) list`
- `balance: reaction`
- `is_balanced: bool`

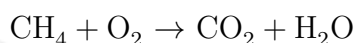
# Chapter VIII

## Exercise 04: Alkane combustion

	Exercise 04
Math, science, history unravelling the mystery...	
Turn-in directory : <i>ex04/</i>	
Files to turn in : *.ml, Makefile	
Forbidden functions : None	
Remarks : n/a	

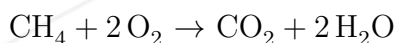
Now that we know what a chemical reaction is, let's do a real one! Do you remember the alkanes I had you write a few exercises ago? Because now we're going to burn them. Our class will be named `alkane_combustion`, and it will be a subtype of `reaction` (what did you expect?).

But first, let's clear up some theory. An alkane combustion means you start with an alkane and some molecular oxygen, and you end up with carbon dioxide and water. Always. That means our reaction will be (for example, with methane):



But wait! It's not actually that simple. I did say you had to have the same exact amount of atoms at the start and at the end, right? And if you look at what I just wrote, that doesn't really add up: you start with 1 carbon, 4 hydrogen and 2 oxygen, and you end with one carbon, 2 hydrogen and 3 oxygen; so you end up with not enough hydrogen and too much oxygen.

That's where **stoichiometrical coefficients** come in. Of course don't forget to look up the word if it sounded like Hebrew to you. But basically they are coefficients you add to the molecules in your reaction, so that you have the same amount of atoms on both sides. As a result:




Now on both sides you have 1 carbon, 4 hydrogen and 4 oxygen. Good, everything works!

Now let's talk about code. You will construct your class with a list of `alkane` objects, and it will implement the `reaction` class's methods:

- **get\_start**: returns the list of molecules at the beginning of the reaction. Throws an exception if the reaction is not balanced.
- **get\_result**: returns the list of molecules at the end of the reaction. Throws an exception if the reaction is not balanced.
- **balance**: returns a new alkane combustion, with the right (and smallest possible) stoichiometrical coefficients so that your combustion is balanced.
- **is\_balanced**: **true** if your reaction is balanced, **false** otherwise.

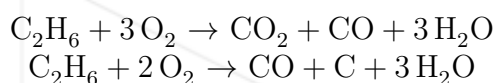
# Chapter IX

## Exercise 05: Incomplete combustion

	Exercise 05
It all started with the Big Bang! (BANG!)	
Turn-in directory : <i>ex05/</i>	
Files to turn in : *.ml, Makefile	
Forbidden functions : None	
Remarks : n/a	

Balancing alkane combustions was the main goal of your day, and if you succeeded in doing that, congratulations! I'm very proud of you. This final exercise is for those of you who want to go a bit further (but you still have to do it if you want full points on the day. :))

You will take your `alkane_combustion` class and add a new method to it, called `get_incomplete_results`. And by incomplete results, I mean incomplete combustion. There are cases when your alkane isn't provided enough oxygen to completely burn, which leads to carbon monoxide or soot being created. For example, with ethane:



Of course, this is only one of the possible outcomes with one ethane. There might be solutions with no carbon dioxide at all, but obviously if there's no carbon monoxide then it's a complete combustion and it doesn't count. Your new method will be able to change the amount of oxygen in the reaction to compute the possible outcomes, which will be returned with type `(int * (molecule * int) list) list`, using the amount of oxygen as the list's index.