1. Getting started with ChocLet

1.1 Launching

To launch the shell of ChocLet, just run the following command .

```
$ java -jar choclet.jar
>
```

1.2 Variable declaration

You can declare variables using the keyword **let**.

```
let x = 10;
let y = "some string";
let z = 13.385;
```

The types of the variables are infered, and you can't change the type of a variable.

```
let x = 10; // x is an int
x = 0.1; // Error, No operator '=' for type 'int' and 'double'
```

ChocLet give some ways to declare arrays.

```
let x = [1, 2, 3]; // x is an array of int, of size 3
let y = [100 of float]; // y is an array of float, of size 100
```

Warning

All the values inside an allocated array, are null.

```
let x = [10 of int];
// let y = x[0] + 10; // NullPointerException
x[0] = 3; // Ok
let y = x[0] + 10; // Ok
```

You can concatenate array of the same type using the operator ' '.

```
let x = [1, 2, 3] ~ [4, 5, 6];
let y = "same for " ~ "strings";
println(x); // [1, 2, 3, 4, 5, 5];
println(y); // same for strings
```

1.3 Source file

You can write source code in a file. The file must have the extension **.clt**. To run a file, there is two ways, the first one is to pass the file as an argument of the jar file.

```
$ java -jar choclet.jar myfile.clt
```

The second one is to import the function described in the file using the keyword **import**. It won't execute the code inside the file, but import all declared function.

```
> import myfile;
```

1.4 Functions

```
def foo (a, b) {
    println (a, b);
}

// foo is a function and we call it
foo (1, "hi !!");
```

1.5 Flow Control

Values can be controlled conditionally using the **if** and **else** statements.

```
let n = 5;
if n < 0 {
    println (n, " is negative");
} else if n > 0 {
    println (n, " is positive");
} else {
    println (n, " is zero");
}
```

You can also do loops with the **while** keyword.

```
let n = 1;
// Loop while n is less than 101
while n < 101 {
    if n % 2 == 0 {
        println ("even");
    } else {
        println ("odd");
    }
    n = n + 1;
}
```

Or iterate over a range of value with the keyword **for**

```
for i in 0 .. 101 {
    if i % 2 == 0 {
        println ("even");
    } else {
        println ("odd");
    }
}
```

2. Choco interface

ChocLet is designed to use Choco solver.

```
let a = choco.int (0, 10);
let b = choco.int (0, 10);
```

This declaration means that **a**, and **b** can have a value between *0* to *10*.

```
(a != b).post ();
```

In this instruction, we inform choco, that we don't want that **a** and **b** have the same value.

```
while choco.solve ()
    println (a, "!=", b);
```

We request a resolution, and while there is a valid solution, we print the value of **a** and **b**.

Choco have some predefined global constraint, and some times we want to use them.

```
in choco {
    // allDifferent is a choco function declared somewhere
    def allDifferent -> ChocoConstraint;
}

// creating an array of 10 var, between 0 and 10
let a = choco.intArray ("A", 10, 0, 10);
choco.allDifferent (a).post (); // Ok
choco.solve ();
println (a);
```

3. Pratical Session

3.1 Question 1

We are a cloud provider. We have already sold 5 VMs to ours clients. Each VM has a cost (between 1 to 10 dollars) for our client (ex [10,3,5,1,9]). Print on the screen our profit.

3.2 Question 2

We have 5 VM. Each VM has a State (0=Off, 1=On) and a cost (between 1 to 10 dollars) for our client. Our data center is very limited, so we cant host all VMs clients. In our new problem, we want exactly *nbRunning* VM On. Write a chocolet code for automatically affecte the State of all VM (depends en *nbRunning*) Print all VM States. (*use the* **sum** *choco constraint*)

3.3 Question 3

Now, we want exactly *nbRunning VM* and also maximize our profit. (profit = the sum of VM On costs). Print our profit. (*use* **maximize()**)

3.4 Question 4

We add the server concept in our data center. Each server has a capacity. Each VM a consumption. (ex : we add 2 servers with 10 and 20 for their capacities. Our 5 VMs have [10,5,3,6,7] for their consumption.) Each VM On, must be placed on a server. (ex : So if the VM1 is placed on Server1, the VM1 will consume all Server1 capacities)

We want exactly *nbRunning* VM On and maximize our profit,

(*use the* **binPacking***choco constraint*)

3.5 Question 5

Each server has a VM power consumption, for each VM placed on it, the server consume Xw. (ex : if the VM power consumption on Server1 is equal to 100, then, if 2 VMs are placed on Server1, the Server1 power consumption is equal to 200) We want exactly *nbRunning* VM On and minimize the total power consumption (*use* **minimize()**)

3.6 Question 6

We want to maximize our profit. Unfortunately, 1W cost 1 dollar. So our profit is the sum of VM On minus the total power consumption. In this question, We want exactly *nbRunning* VM On and maximize our profit.

3.7 Question 7

We want optimize our profit. So, when a server is unused (no VM placed on it), the server is turned off. (so the power consumption of this server = 0). When a server is On, his power consumption depends on VMs running on it. More precisely, depends of the number of unit consumption. (ex : the Server1 has a unit consumption of 5W (5W by capacity). Just one VM is running on the

Server 1. The VM has a consumption of 5, then the total Server1 power consumption is then 25W.)
We want exactly *nbRunning* VM On and maximize our profit. (*use the* **count** *choco constraint*)