Basic MiniZinc: Parameters and Variables

Parameters: input data for the problem, constants

```
int: n;
n = 3;
% equivalent to:
int: n = 3;
par int: n = 3;
% or with bounded domains
0..10: m; % a fixed value (input by a user) between 0 and 10
% set types
set of int: AGENTS = 1..n; % {1, ..., n}
```

Decision Variables: variables to be assigned by a solver

```
var int: n;
var 0..10: x;
% immediate declaration (useful for dependent expressions)
var int: y = 2*x;
```


Basic MiniZinc: Arrays and Sets

Arrays: for parameters/variables of different sizes

```
int: n;
array[1..n] of int: height; % height[i] denotes the height of truck i
% also for decisions
array[1..n] of var bool: x; % x[i] holds if we select item i
% typical usage
set of int: ITEMS = 1..n;
array[ITEMS] of var bool: x;
```

Sets: set type for parameters and decisions (only set of int/subtype int)

```
set of int: AGENTS = 1..n;
set of int: CAP = 1..m;
array[AGENTS] of set of CAP: offers; % [{1}, {2}, {1,2}]

var set of AGENT: selectedTeam;
```


Basic MiniZinc: Types

Allowed primitive types for variables/parameters are

- Integer `int` or subtypes:
  - range `1..n`
  - explicit set {1,5,7}
- Floating point number `float` or subtypes:
  - range `0.0 .. 1.0`
  - explicit set {1.0, 2.5, 3.7}
- Boolean `bool`
- Arrays, Sets

Constraints, Objective and Output

Constraints: to restrict assignments

```
var 0..10: x; var 0..10: y; var 0..10: z;
constraint x + y = z;
constraint x mod 2 = 0;
% 'forall' concept for arrays
par int: n; array[1..n] of var 0..10: t;
% each t[i] should be even:
constraint forall(i in 1..n) (t[i] mod 2 = 0);
```

Solve Item: provides the objective (one per model)

```
solve satisfy; % for a satisfaction problem
solve maximize sum(i in 1..n) (t[i]);
solve minimize x + y;
```

String Output: one output item per model

```
var 0..10: x;
output ["The value of x is: \(x)"]
```