Checklist for Designing or Reading a Model

1. Each array index occurs twice in the comment on the array's declaration; ex.: `array[I,J] of ...: X; % X[i,j] = the ... i ... j`
2. Each index range of an array either starts from `1` or is an `enum`, for clarity
3. Beware of decision variables declared without tight domains
4. No explicit decision variables of type `opt` τ are used (in this course)
5. No `sum|forall(i in 1..x)` with a decision variable `x` is used
6. Beware of `where` θ and `if` θ with test θ containing decision variables
7. Beware of explicit (`<->`) and implicit (`bool2int(...)`) reification
8. Beware of logical negation and disjunction: `not`, `\/`, `exists`, `xor`, `xorall`, `if` θ `then` ϕ `else` ψ `endif`, `<-`, `->`, `<->`
9. Beware of nonlinear, `pow`, `div`, `mod` constraints on decision variables

Checklist for Designing or Reading a Model

Motivation
all_different
nvalue
global_cardinality
element
bin_packing, knapsack
cumulative, disjunctive
circuit, subcircuit
lex_lesseq
regular, table
Checklist

10 The constraint predicates with the most specific meanings are used

11 Global constraints are not replaced by their definitions

12 Constraints over shared decision variables are ideally merged

13 The `element` predicate is not used explicitly, for clarity

14 Each function on small sets is encoded by an implicit `element` if need be

15 Each relation over small sets is encoded by `regular` or `table` if that is faster than a formulation in the scope of checklist items 6 to 9 (of Topic 2)

Conventions of all Slides (recommended!)

- Scalar identifiers (`bool`, `enum` items, `int`) start with a lowercase letter.
- Mass identifiers (`array`, `enum`, `set`) start with an uppercase letter.
- Arrays have self-explanatory function identifiers: a given|unknown total function $f\colon X \to Y$ can be modelled as `array[X] of par|var Y: F`.
- Index identifiers are lowercase and mnemonic: memory aid.
- Comments about the *next* line end in "`:`", like line 2 in the example below.

Example

```
1 int: nQueens; % the given number of queens
2 % Row[c] = the row number of the queen in column c:
3 array[1..nQueens] of var 1..nQueens: Row;
```

Variable `Row[c]` is like *Row*(*c*), denoting the function *Row* applied to arg. *c*. The array `Row` is *not* a variable, but an *array of* variables: it has row numbers, but calling it `Rows` would make `Rows[c]` seem to denote a *set* of rows for `c`!

Ideas for Debugging and Accelerating a Model

- If there are no solutions (or missing solutions) to a known-to-be satisfiable instance, then:
  - Comment away constraints in order to increase the solution set and thereby find unsatisfiable constraints.
  - In the IDE or CLI, choose findMUS as the backend in order to find a minimal unsatisfiable subset (MUS) of the constraints: see Section 3.8 of the MiniZinc Handbook.
- In the IDE, choose “Run $>$ Profile compilation” in order to see per model line the numbers of constraints and decision variables generated by its flattening, and the flattening time: if some of these numbers are extreme, then you probably ran afoul of items of the checklist on the next slide.
- In the IDE, choose “Run $>$ Compile” in order to inspect the flat code.