

# Cheat sheet Ortools

## Basics

```
from ortools.sat.python import cp_model
```

## Creation of a model

```
model = cp_model.CpModel()
```

## Variables

### - Booleans

```
X = cp_model.NewBoolVar(name="X")
X.Not() or ~X is the negation of X
[New API adding : create vector of variable]
X = model.NewBoolVarSeries("X", pd.Series(range(N)).index)
```

### - Integers

```
X = cp_model.NewIntVar(lb=lb, ub=ub,
                        name="X")
[New API adding : create vector of variable]
X = cp_model.NewIntVarSeries(name="X",
                              index=pd.Series(range(N)).index,
                              lower_bounds=df["lower_bounds"],
                              upper_bounds=df["upper_bounds"])
```

Advanced : Initialisation from Domain :  
If we know some forbidden/allowed values of our integer variable, better use "Domain" functionalities !

```
X = cp_model.NewIntVarFromDomain(
    cp_model.Domain.FromValues([list_values]),
    name="X")
```

```
X = cp_model.NewIntVarFromDomain(
    cp_model.Domain.FromIntervals([[1, 2], [4, 6]]),
    name="X")
```

### - Intervals

## Non optional

```
start=cp_model.NewIntVar(...) or int or Constant
end=cp_model.NewIntVar(...) or int or Constant
size=cp_model.NewIntVar(...) or int or Constant
X=cp_model.NewIntervalVar(start=start, sizes=size, end=end, name="X")
```

```
[Multiple =]
X=cp_model.NewIntervalVarSeries(name="X",
                                index=df.index,
                                starts=start, sizes=size, ends=end,
                                name="X")
```

## Optional

```
start=cp_model.NewIntVar(...) or int or Constant
end=cp_model.NewIntVar(...) or int or Constant
size=cp_model.NewIntVar(...) or int or Constant
is_present=cp_model.NewBoolVar(...) or bool
X=cp_model.NewOptionalIntervalVar(start=start, sizes=size, end=end,
                                   is_present=is_present, name="X")
```

```
[Multiple =]
X=cp_model.NewOptionalIntervalVarSeries(name="X",
                                         index=df.index,
                                         starts=starts,
                                         sizes=sizes, ends=ends,
                                         are_present=are_present,
                                         name="X")
```

## Boolean constraints

```
model.AddBoolAnd(X,Y,...) : All boolean variables are True
model.AddBoolOr(X, Y, ...) : At least on boolean is True
model.AddBoolXor(X, Y...) : Exactly 1 boolean is true
model.AddAtLeastOne(X, Y...) : same ad AddBoolOr
model.AddAtMostOne(X, Y...) : sum(literals)<=1
model.AddExactlyOne(X, Y...) : sum(literals)==1 (mathematically like Xor)
```

## Linear constraints :

```
model.Add(X+2*Y<=C)
Advanced syntax :
1) model.AddLinearConstraint(X+2*X, lb=l, ub=u)
   Same as
   model.Add(X+2*Y<=u)
   model.Add(X+2*Y>=b)

2) model.AddLinearExpressionInDomain(X+2*X,
   domain=cp_model.Domain.FromValues(...))
   Constraint the value of the linear expression to be a given domain
(see)
```

## Global constraints :

### On integers :

```
- model.AddAllDifferent(*X) : force different values to a list of variable
- model.AddReservoirConstraint(times,
                               level_changes,
                               min_level,
                               max_level)
```

Reservoir start at level 0, and then for every times[i], it evolves with +level\_changes[i]. The constraint insure to stay between min\_level and max\_level.

```
- AddReservoirConstraintWithActive(
    times, level_changes, actives,
    min_level,max_level
)
```

Same as previous the level change are conditioned to the "actives" variable.

### On Interval

```
- model.AddNoOverlap(*X (list of interval or optional interval)) :
No overlapping between any task in the list.
- model.AddCumulative(intervals=*X, demands=demands,
                      capacity=capacity)
```

If each interval is consuming a given demand of resource then the cumulated consumed value at any time can't exceed the capacity  
Note that for those 2 global constraint, the non-present interval (optional interval with a flag == False) are ignored, which makes this very convenient !!)

Other utilities constraint

```
model.AddElement(index (var),
                 variables (list of var),
                 target (var))
set target=variables[index]
```

```
model.AddMapDomain(var (intvar), bool_var_array, offset):
Add constraint bool_var_array[var-offset]==True
Can be used to map a list boolean representation to an integer one.
```