# Cheatsheet JuMP

## Grundlagen Operations Research

Kompakte Referenz aller JuMP-Bausteine, die in diesem Kurs vorkommen. Schritt-für-Schritt-Einführung: Pakete & JuMP.

## Grundgerüst

```
using JuMP, HiGHS

modell = Model(HiGHS.Optimizer)
set_silent(modell)          # Solver-Ausgabe abschalten

# ... Variablen, Zielfunktion, Nebenbedingungen ...

optimize!(modell)           # lösen
```

## Variablen

### Deklaration

```
@variable(modell, X)                # frei (unbeschränkt)
@variable(modell, X >= 0)           # nichtnegativ
@variable(modell, 0 <= X <= 10)     # mit beiden Schranken

@variable(modell, X >= 0, Int)      # ganzzahlig
@variable(modell, X, Bin)           # binär (0 oder 1)
```

### Container

```
@variable(modell, X[1:5] >= 0)          # Vektor von Variablen
@variable(modell, X[1:3, 1:4] >= 0)     # Matrix (z. B. Transport)
@variable(modell, X[t in T], Bin)       # über Indexmenge T

X[2]        # einzelne Variable
X[1, 3]     # Element einer Variablenmatrix
```

## Zielfunktion

```
@objective(modell, Max, 20X1 + 30X2)            # maximieren
@objective(modell, Min, 4X1 + X2)               # minimieren
@objective(modell, Min, sum(c[i] * X[i] for i in 1:5))
```

## Nebenbedingungen

```
# Benannt (empfohlen, der Name erlaubt später shadow_price etc.):
@constraint(modell, Paletten, X1 + X2 <= 5)
@constraint(modell, Bedarf, X1 + 3X2 >= 9)
@constraint(modell, Bilanz, X1 - X2 == 0)

# Eine Bedingung je Index (Container):
@constraint(modell, Kapazitaet[i in 1:3],
    sum(X[i, j] for j in 1:4) <= a[i]
)

# Summen mit Bedingung:
@constraint(modell, Fluss,
    sum(X[k] for k in 1:9 if von[k] == 1) == 1
)
```

## Lösen und Ergebnisse

```
optimize!(modell)

termination_status(modell)   # IMMER zuerst prüfen!
objective_value(modell)      # optimaler Zielfunktionswert F*
value(X1)                    # Wert einer Variablen
value.(X)                    # Werte eines Containers (Punkt!)
```

### Statuswerte (vgl. Vorlesung 5)

```
termination_status(modell) == OPTIMAL           # optimale Lösung
termination_status(modell) == INFEASIBLE        # unzulässig
termination_status(modell) == DUAL_INFEASIBLE   # unbeschränkt
```

## Sensitivität (vgl. Vorlesungen 3 und 4)

```
shadow_price(Paletten)   # Verbesserung von F* bei Relaxation
                         # der Nebenbedingung um eine Einheit
reduced_cost(X1)         # reduzierte Kosten der Variablen
```

> ⚠ Warnung
>
> Vorzeichen-Konvention: `shadow_price` misst die **Verbesserung** von $F^*$ bei Relaxation der Nebenbedingung. Bei Maximierungsmodellen ist das identisch mit dem Simplex-Multiplikator (der Dualvariablen) aus der Vorlesung. Bei **Minimierungsmodellen** ist das Vorzeichen gegenüber der klassischen Dualvariablen bei ≥-Nebenbedingungen stets und bei Gleichungen teilweise gedreht. Vergleichen Sie dort im Zweifel mit der Handrechnung.

## Modell nachträglich ändern

```
fix(X1, 9; force = true)     # Variable auf Wert festsetzen
unfix(X1)                    # Festsetzung aufheben

set_lower_bound(X1, 2)       # untere Schranke setzen/ändern
set_upper_bound(X1, 8)       # obere Schranke setzen/ändern
delete_upper_bound(X1)       # obere Schranke entfernen

set_integer(X1)              # Ganzzahligkeit ergänzen
unset_integer(X1)            # ... wieder entfernen

# danach einfach erneut lösen:
optimize!(modell)
```

## Modell prüfen

```
print(modell)                # Modell wie auf dem Papier anzeigen
num_variables(modell)        # Anzahl Variablen
solution_summary(modell)     # Zusammenfassung nach dem Lösen
```

## Wiederkehrende Ausgabezeile des Kurses

```
println("F* = ", objective_value(modell),
        " mit X = ", value.(X))
```

> Tipp
>
> Julia-Grundlagen (Vektoren, Schleifen, Broadcasting) stehen im Cheatsheet Julia.

## Bibliografie