# Term-Prädikate

## Übersicht

- Terme klassifizieren
- `=.., functor, arg`
- Terme vergleichen und ordnen

## Ziel

- Selten gebrauchte Prädikate
  - werden mehr der Vollständigkeit halber vorgestellt
  - → kein Prüfungsstoff

# Prolog-Prädikate zur Klassifikation

**Das Prädikat atomic/1 besagt, ob ein Term atomar (Atom oder Zahl) ist.**

```
?- atomic(fido).
yes
```

```
?- atomic(hund(fido)).
no
```

**Das Prädikat var/1 besagt, ob ein Term eine ungebundene Variable ist.**

```
?- var(X).
yes
```

```
?- var(fido).
no
```

# =../2

**Mit =.. kann ein Term in eine Liste verwandelt werden,**

- deren erstes Element gleich dem Funktor des Terms ist,
- und deren restliche Elemente gleich den einzelnen Argumenten des Terms sind.

**Wie viele andere Prädikate ist auch =.. umkehrbar.**

```
?- f(schoen, gut) =.. Liste.
Liste = [f, schoen, gut]
```

```
?- Term =.. [f, schoen, gut].
Term = f(schoen, gut)
```

# functor/3

**functor(Term, F, S) gelingt, wenn**

- Term ist zusammengesetzter Term mit Funktor F und Stelligkeit S,
- oder Term ist ein Atom/eine Zahl, wobei Term = F und S = 0

```
?- functor(hund(fido), Funktor, Stelligkeit).
Funktor = hund, Stelligkeit = 1
```

```
?- functor(a, F, S).
F = a, S = 0
```

```
?- functor(b(c, d), b, 3).
no
```

```
?- functor(Neuer_Term, b, 1).
Neuer_Term = b(_22)
```

# arg/3

**arg(N, Term, Arg) gelingt, wenn Arg das N-te Argument von Term ist.**

- Weder N noch Term dürfen ungebundene Variablen sein.

```
?- arg(3, f(a, b, c), Was).
Was = c
```

```
?- arg(1, f(a, b, c), a).
yes
```

```
?- arg(2, f(a, b, c), d).
no
```

# copy_term/2

## copy_term(Term1, Term2)

- erzeugt eine Kopie von Term1
- ersetzt alle ungebundenen Variablen in der Kopie durch frische
- unifiziert das Ergebnis mit Term2

```
?- copy_term(f(X, Y, X), Neu).
Neu = f(_1, _2, _1)
```

# Terme vergleichen

==/2 gelingt, wenn zwei Terme gleich sind. Die Terme werden dabei nicht unifiziert, Variablen nicht gebunden.

```
?- hund(f) == hund(f).
yes
```

```
?- hund(f) == hund(X).
no
```

\==/2 gelingt bei Ungleichheit zweier Terme. Die Terme werden dabei nicht unifiziert, Variablen nicht gebunden.

```
?- hund(f) \== hund(f).
no
```

```
?- hund(f) \== hund(X).
yes
```

# Terme ordnen

**@</2 gelingt, wenn der erste Term in alphabetischer Ordnung vor dem zweiten steht.**

```
?- adam @< apfel.
yes
```

```
?- adam @< adam.
no
```

**Weiterhin sind folgende Operatoren definiert, um Terme zu ordnen:**

- @>
- @=<
- @>=