# Interne Funktion und named let

Interne Funktion
und
named let

Erläuterung am Beispiel

# Interne Funktion und named let

- Zum Projekt
  TSP mit Genetischen Algorithmen
  gibt es zwei Versionen:
  - In einer Version wird durchgehend mit **let** und **named let** gearbeitet.
  - In der anderen Version werden nach Möglichkeit interne Funktionen mit **define** verwendet.
- Beispielhaft erläutere ich am Beispiel der Funktion **erzeuge-Population**.

# Interne Funktion und named let

```
(define
 (erzeuge-Population Anzahl)
 (define
  (erzeuge-intern Population n)
  (cond
   ((zero? n)
    Population)
   (else
    (erzeuge-intern
     (cons
      (erzeuge-Individuum)
      Population)
     (sub1 n)))))
 (erzeuge-intern '() Anzahl)
 )
```

innere Funktion

# Interne Funktion und named let

- Für beide Varianten gelten gemeinsame Eigenschaften:
  - Sie gehören zur Umgebung (*scope*) der Funktion, in die sie eingebettet sind und
  - können daher auf deren Parameter zugreifen.
  - Dennoch überdecken bei Namensgleichheit interne Bindungen die äußeren.

# Interne Funktion und named let

```
(define
 (erzeuge-Population Anzahl)
 (define
  (erzeuge-intern Population Anzahl)
  (cond
   ((zero? Anzahl)
    Population)
   (else
    (erzeuge-intern
     (cons
      (erzeuge-Individuum)
      Population)
     (sub1 Anzahl)))))
 (erzeuge-intern '() Anzahl)
 )
```

wäre zulässig und nur interne **Anzahl** wird bei der Rekursion verändert

# Interne Funktion und named let

- Grundsätzliche Aufgabe von **let** ist das Binden von Namen für den nachfolgenden Auswertungsteil.
- Im folgenden primitiven Beispiel wird **n** intern an das Ergebnis der Eingabe gebunden und im Auswertungsteil um drei erhöht zurück gegeben.

```
(let ((n (read))) (+ n 3))
Eingabe war: 5
Ausgabe ist: 8
```

# Interne Funktion und named let

```
(define
 (erzeuge-Population Anzahl)
 (let
    loop
  ((Population ())
   (n Anzahl))
  (cond
   ((zero? n)
    Population)
   (else
    (loop
     (cons
      (erzeuge-Individuum)
      Population)
     (sub1 n))))))
```

named let

# Interne Funktion und named let

- Bei einem **named let** folgt auf das **let** ein Name, hier **loop**, der lokal als Funktionsname behandelt wird.

```
(let
  loop
 ((Population ())
  (n Anzahl))
 (cond
  ...
  (else
   (loop
    (cons (erzeuge-Individuum) Population)
    (sub1 n))))))
```

Parameter der umgebenden Funktion: **Anzahl**

lokaler Parameter **Population**

lokaler Parameter **n**