

# HtDP-TL Stichwortverzeichnis

19. März 2020

Akkumulatoren	Zweck	<ul style="list-style-type: none"> <li>• Zwischenspeicher bei rekursiven Funktionen</li> </ul>	4B/128
	Verwendung	<ul style="list-style-type: none"> <li>• Rekursiver Aufruf einer Hilfsfunktion mit zusammengesetzter Funktion als Parameter</li> <li>• z.B.: ...(<code>counter (+ accu 1)</code>)...</li> </ul>	4B/128

Arithmetrische Operationen	Operationen	<ul style="list-style-type: none"> <li>• <code>( + . . ) ( - . . ) ( * . . ) ( / . . ) ( modulo . . )</code></li> </ul>	4A/66
	Operanden	<ul style="list-style-type: none"> <li>• atomar oder zusammengesetzt möglich</li> </ul>	4A/70
	mehr als 2 Operanden	<ul style="list-style-type: none"> <li>• z.B.: <code>( + 1 2 3 4 )</code></li> <li>• Jedoch bei <code>-</code> und <code>/</code> → erste Zahl Minuend bzw. Dividend</li> </ul>	4A/73
	nicht exaktes Ergebnis	<ul style="list-style-type: none"> <li>• Darstellung der Zahl mit <code>#i...</code></li> </ul>	4A/75
	Operationen	<ul style="list-style-type: none"> <li>• auf ganzen Zahlen: <ul style="list-style-type: none"> <li>• <code>( floor . ) ( ceiling . ) ( gcd . . ) ( modulo . . )</code></li> </ul> </li> </ul>	4A/79

Boolsche Operationen	Literale	<ul style="list-style-type: none"> <li>• <code>#t</code> → true   <code>#f</code> → false</li> </ul>	4A/86
	Verknüpfung	<ul style="list-style-type: none"> <li>• <code>( and . . . ) ( or . . . ) ( not . )</code></li> </ul>	4A/87
	Vergleiche	<ul style="list-style-type: none"> <li>• <code>( = . . . ) ( &lt; . . . ) ( &lt;= . . . )</code> → Vergleich von zwei aufeinanderfolgenden Parametern</li> </ul>	4A/89
	Verzweigung(if)	<ul style="list-style-type: none"> <li>• <code>( if ( Bedingung ) (falls true) (falls false) )</code></li> </ul>	4A/92
	Funktionen	<ul style="list-style-type: none"> <li>• <code>integer?</code> → <code>#t</code> falls Parameter ganze Zahl</li> <li>• <code>number?</code> → <code>#t</code> falls Parameter Zahl</li> <li>• <code>real?</code> → <code>#t</code> falls Imaginärteil exakt 0</li> <li>• <code>rational</code> → <code>#t</code> falls Parameter rational ist</li> <li>• <code>natural</code> → <code>#t</code> falls Parameter natürliche Zahl</li> <li>• <code>string?</code> → <code>#t</code> falls Parameter String</li> <li>• <code>my-struct?</code> → <code>#t</code> falls Element von Klasse <code>my-struct</code></li> </ul>	4A/93

Funktion	Syntax	<ul style="list-style-type: none"> <li>• <code>( define ( my-function Parameter ) ( Anweisung ) )</code> → Präfixnotation (zuerst der Operand, dann Operanden)</li> </ul>	4A/43
	Aufruf	<ul style="list-style-type: none"> <li>• <code>( my-function Parameter )</code></li> </ul>	4A/49
	Verstecken von Definitionen	<ul style="list-style-type: none"> <li>• <code>( local ... )</code> <ul style="list-style-type: none"> <li>• nur letzter Ausdruck Wert des local Ausdrucks</li> <li>• Erstellung von lokalen Definitionen</li> </ul> </li> </ul>	4A/135
	cond	<ul style="list-style-type: none"> <li>• Syntax: <code>( cond [ (Bedingung 1) Folge 1 ] [ (Bedingung 2) Folge 2 ] [ else ... ] )</code></li> </ul>	4B/61
	Funktionen als Daten	<ul style="list-style-type: none"> <li>• z.B.: <code>( define add + )</code> → <code>add</code> ist Konstante vom Typ aller Funktionen, die zwei Zahlen als Parameter haben und eine Zahl zurückliefern → ;; Type: <code>number number → number</code> <ul style="list-style-type: none"> <li>• <code>( add 2 3 ) → 5</code></li> </ul> </li> </ul>	4C/9

	Höherer Ordnung	<ul style="list-style-type: none"> <li>Funktionen, die Funktionen als Parameter oder Rückgabe haben</li> <li>Vertrag: z.B.: (number <math>\rightarrow</math> number) number <math>\rightarrow</math> number</li> </ul>	4C/18
	Aufweichung	<ul style="list-style-type: none"> <li>Zeitliche Abläufe in Racket:</li> <li>#&lt;void&gt; als Wert <math>\rightarrow</math> Konstante mit leerem Wert</li> <li>(begin ...) <math>\rightarrow</math> Block, der mehrere Anweisungen enthalten kann <math>\rightarrow</math> Letzter Ausdruck = Rückgabewert</li> <li>(set! my-constant value) <math>\rightarrow</math> Überschreibt Konstante</li> </ul>	4D/56

Funktionales Programmieren	Funktionen	<ul style="list-style-type: none"> <li>zentrale Bausteine des funktionalen Programmierens</li> <li>Zerlegung der zu erstellenden Funktionalität in Funktionen</li> </ul>	4A/23
	Deklaratives Programmieren	<ul style="list-style-type: none"> <li>Nur Verwendung der Formel des Ergebnis <math>\rightarrow</math> keine zeitlichen Abläufe <math>\rightarrow</math> keine Vorstellung vom Computerspeicher</li> </ul>	4A/31
	referentielle Transparenz	<ul style="list-style-type: none"> <li>Einziger Effekt von Funktionen ist der Rückgabebetyp <math>\rightarrow</math> void Methoden sinnlos</li> </ul>	4A/32

Klammern	atomar	<ul style="list-style-type: none"> <li>Einzelne Bestandteile müssen nicht geklammert werden (atomar)</li> </ul>	4A/69
	Zusammengesetzter Ausdruck	<ul style="list-style-type: none"> <li>Klammern sind notwendig</li> </ul>	4A/69
	Notwendig	<ul style="list-style-type: none"> <li>Keine Punkt vor Strich Rechnung <math>\rightarrow</math> Klammern notwendig</li> </ul>	4A/70

Kommentare	Einzeilig	<ul style="list-style-type: none"> <li>; ...</li> </ul>	4A/56
	Funktion	<ul style="list-style-type: none"> <li>Festlegung der Eingabeparameter und Ausgabeparameter <math>\rightarrow</math> Bsp.: ;; Type: number number <math>\rightarrow</math> number ;; Returns: ... <math>\rightarrow</math> Vertrag der Funktion</li> <li>;; Precondition: ... <math>\rightarrow</math> siehe Laufzeitchecks</li> <li>höhere Ordnung: z.B. (number <math>\rightarrow</math> number) number <math>\rightarrow</math> number</li> </ul>	4A/106
	iff	<ul style="list-style-type: none"> <li>gängige Abkürzung für 'if, and only if,...'</li> </ul>	4B/75

Konstanten	Häufigkeit	<ul style="list-style-type: none"> <li>nur Konstanten in Racket</li> </ul>	4A/33
	Syntax	<ul style="list-style-type: none"> <li>( define myConstant Wert )</li> <li>zusammengesetzter Ausdruck <math>\rightarrow</math> Wert in Klammern</li> </ul>	4A/53

Konventionen	Identifer	<ul style="list-style-type: none"> <li>Keine Großbuchstaben</li> <li>Bindestriche zwischen Wörtern <math>\rightarrow</math> my-identifer</li> </ul>	4A/58
	Kommentare	<ul style="list-style-type: none"> <li>einzeilig: ;; statt ; (bessere Sichtbarkeit)</li> </ul>	

Laufzeitchecks	Funktion	<ul style="list-style-type: none"> <li>Unterbrechen des Programms und Testen der Funktion <ul style="list-style-type: none"> <li>falls true <math>\rightarrow</math> Programm wird weitergeführt</li> <li>falls false <math>\rightarrow</math> Abbruch des Programms mit Fehlermeldung</li> </ul> </li> </ul>	4A/112
	check-expect	<ul style="list-style-type: none"> <li>( check-expect ( divide 15 3 ) 5 ) <ul style="list-style-type: none"> <li>Test ob <math>15/3 = 5</math></li> </ul> </li> </ul>	4A/112
	check-within	<ul style="list-style-type: none"> <li>( check-within ( divide pi e ) 1.15 0.01 ) <ul style="list-style-type: none"> <li>Test ob pi/e weniger als 0.01 von 1.15 abweicht</li> </ul> </li> </ul>	4A/113
	check-error	<ul style="list-style-type: none"> <li>( check-error ( divide 15 0 ) “/: division by zero“ ) <ul style="list-style-type: none"> <li>Überprüfen ob Fehlermeldungen gleich (Dokumentation)</li> <li>Fehlermeldungen sind Strings</li> </ul> </li> </ul>	4A/114

	In Funktion	• z.B.: Division ( if ( = y 0 ) ( error "Divison by 0" ) ( / x y ) )	4A/117
Lambda-Ausdrücke	Syntax	• (lambda (x y) (+(* x x)(* y y)))	4C/99
	Beispiel	<ul style="list-style-type: none"> <li>• ;; Type: (number -&gt; number) (number -&gt; number)</li> <li>• ;; -&gt; (number number -&gt; number)</li> <li>• (define ( name fct1 fct2 )  (lambda (x,y) (+ (fct1 x) (fct2 y))))  → zurückgelieferter Wert auch Funktion</li> </ul>	4C/105
	Beispiel Verwendung	<ul style="list-style-type: none"> <li>• ( (name bspfct1 bspfct2) param1 param2))</li> <li>→ Return aus Funktion name" wird danach mit param1 und param2 aufgerufen</li> <li>→ Anwendung von bspfct1 auf param1</li> </ul>	4C/108
	filter	<ul style="list-style-type: none"> <li>• Filterung abhängig von Prädikat</li> <li>• Beispiel: (define (my-filter prädikat liste)...) <ul style="list-style-type: none"> <li>• (my-filter (lambda(x) (&lt; x 10)) liste)</li> </ul> </li> <li>• Durch Verwendung von Lambda → variable Filter</li> </ul>	4C/110
	map	<ul style="list-style-type: none"> <li>• Anwendung einer Funktion auf jedes Element der Liste</li> <li>• Beispiel: (define (my-map fct list)...) <ul style="list-style-type: none"> <li>• (my-map (lambda(x) (* x 10)) list)</li> </ul> </li> <li>• Durch Verwendung von Lambda → variable Anpassung</li> </ul>	4C/125
	fold	<ul style="list-style-type: none"> <li>• Zusammenrechnen einer Liste nach Vorschrift <ul style="list-style-type: none"> <li>• foldr von rechts, foldl von links</li> </ul> </li> <li>• Beispiel: (define (my-fold init fct list)...) <ul style="list-style-type: none"> <li>• (my-fold 0 (lambda(x,y) (+ x y)) list)</li> </ul> </li> <li>• Durch Verwendung von Lambda → variable Verrechnung</li> </ul>	4C/139

Listen	Syntax	• ( define my-list ( list 1 2 3 ) )	4B/3
	cons	<ul style="list-style-type: none"> <li>• Anhängen an Liste</li> <li>• z.B.: ( define my-list2 ( cons 0 list1 ) )  → 0 1 2 3</li> </ul>	4B/6
	empty	• leere Liste (z.B.: (... ( cons 0 empty ) ) )	4B/8
	Funktionen	<ul style="list-style-type: none"> <li>• ( first my-list ) → erstes Element</li> <li>• ( rest my-list ) → Alle Elemente außer Erstes</li> </ul>	4B/9
	Vertrag	<ul style="list-style-type: none"> <li>• ;; Type: ( list of ... ) → ...</li> <li>• ;; Type: ( list of ANY ) → ... , falls Typ beliebig</li> <li>• ;; Type: X → X , Eingabetyp beliebig, Ausgabe vom selben Typ</li> <li>• ;; Type: ANY → ANY , Eingabetyp beliebig, Ausgabebetyp auch</li> </ul>	4B/16
	Rekursion	<ul style="list-style-type: none"> <li>• Beispiel Summe:  → ( if (empty? list ) 0 ( + ( first list ) ( sum ( rest list ) ) ) )</li> <li>• Beispiel Liste als Ausgabe:  → .. ( if ( empty? list ) empty ( cons ( sqrt ( first list ) ( sqrts ( rest list ) ) ) ) )</li> </ul>	4B/41
	Filter	<ul style="list-style-type: none"> <li>• Aussortieren von gewissen Elementenn <ul style="list-style-type: none"> <li>• ( define ( filter list ) (if (Sortierbedingung)  → ( cons ( first list ) ( filter list ) ) , falls #t  → ( filter list ) ) ) , falls #f</li> </ul> </li> </ul>	4B/42
	homogen heterogen	<ul style="list-style-type: none"> <li>• Listen auch mit unterschiedlichen Typen möglich → heterogen</li> <li>→ ;; Type: ( list of ANY )...</li> </ul>	4B/107

Objektmodell	nur Konstanten	→ Werte werden immer kopiert	4A/126
	Idealisiert	• Abweichung vom Modell im Hintergrund zur Optimierung	4A/129

Rekursion	Zweck	<ul style="list-style-type: none"> <li>• grundlegendes Konzept zur Steuerung des Programmablaufs</li> </ul>	4A/161
-----------	-------	---	--------

  

Streams	Eigenschaften	<ul style="list-style-type: none"> <li>• nicht in DrRacket (HtDP-TL), sondern nur in Racket</li> <li>• Variante von Listen, ziemlich ähnlich</li> <li>• Einziger Unterschied: Elemente müssen nicht unbedingt physisch existieren</li> </ul>	8/122
	Funktionen	<ul style="list-style-type: none"> <li>• (stream-cons x str): Hängt x vorne an str</li> <li>• (stream-first str): Gibt erstes Element von str zurück</li> <li>• (stream-rest str): Stream ohne erstes Element</li> <li>• (stream-empty? str): Abfrage ob Stream leer ist</li> <li>• Map, Filter und fold analog zu Listen</li> </ul>	8/129

  

Struct	Zweck	<ul style="list-style-type: none"> <li>• Zusammenfassung von Elementen → Vergleichbar mit Klasse mit public Attributen aber ohne Methoden</li> </ul>	4B/94
	Syntax	<ul style="list-style-type: none"> <li>• ( define-struct my-struct ( attribut1 attribut2 )) → Attribute werden auch 'Felder' genannt</li> </ul>	4B/99
	'Konstruktor'	<ul style="list-style-type: none"> <li>• ( define my-struct-name ( make-my-struct wert1 wert2 ))</li> </ul>	4B/101
	Attributzugriff	<ul style="list-style-type: none"> <li>• .. ( my-struct-attribut1 my-struct-name ) ..</li> </ul>	4B/104
	mystruct?	<ul style="list-style-type: none"> <li>• #t falls Element von Typ my-struct</li> </ul>	4B/112
	Funktionen als Attribute	<ul style="list-style-type: none"> <li>• ( define-struct functions ( fct1 fct2 ))  <ul style="list-style-type: none"> <li>• z.B.: ( define a ( make-functions + * )) → fct1 von a ist Summe, fct2 von a ist Multiplikation</li> </ul> </li> </ul>	4C/11

  

Symbole	Syntax	<ul style="list-style-type: none"> <li>• ( define my-symbol 'helloworld )</li> </ul>	4A/83
	Verwendung	<ul style="list-style-type: none"> <li>• Symbole stehen nur für sich selbst → keine Bedeutung für Racket</li> </ul>	4A/83
	Gleichheit	<ul style="list-style-type: none"> <li>• ( if (symbol=? my-symbol1 my-symbol2) (falls #t) (falls #f) )</li> </ul>	4A/84

  

Zahlen	Exakte Zahlen	<ul style="list-style-type: none"> <li>• ganzzahlig (z.B.: 5)</li> <li>• rational (z.B.: <math>3/5</math> → wird auch als Bruch gespeichert)</li> </ul>	4A/61
	Nichtexakte Zahlen	<ul style="list-style-type: none"> <li>• irrationale Zahlen (z.B.: ( sqrt 2 )</li> </ul>	4A/62
	Komplexe Zahlen	<ul style="list-style-type: none"> <li>• komplexe Zahlen <ul style="list-style-type: none"> <li>• Realteil + Imaginärteil (i nach Imaginärteil)</li> </ul> </li> </ul>	4A/63

  

		•	
		•	
		•	
		•	
		•	