

HtDP-TL Stichwortverzeichnis

20. Januar 2020

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

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

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

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

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

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

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

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

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

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

Laufzeitchecks	Funktion	<ul style="list-style-type: none"> Unterbrechen des Programms und Testen der Funktion <ul style="list-style-type: none"> falls true \rightarrow Programm wird weitergeführt falls false \rightarrow Abbruch des Programms mit Fehlermeldung 	4A/112
	check-expect	<ul style="list-style-type: none"> (check-expect (divide 15 3) 5) <ul style="list-style-type: none"> Test ob $15/3 = 5$ 	4A/112
	check-within	<ul style="list-style-type: none"> (check-within (divide pi e) 1.15 0.01) <ul style="list-style-type: none"> Test ob pi/e weniger als 0.01 von 1.15 abweicht 	4A/113
	check-error	<ul style="list-style-type: none"> (check-error (divide 15 0) “/: division by zero“) <ul style="list-style-type: none"> Überprüfen ob Fehlermeldungen gleich (Dokumentation) Fehlermeldungen sind Strings 	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"> • ;; Type: (number -> number) (number -> number) • ;; -> (number number -> number) • (define (name fct1 fct2) (lambda (x,y) (+ (fct1 x) (fct2 y)))) → zurückgelieferter Wert auch Funktion 	4C/105
	Beispiel Verwendung	<ul style="list-style-type: none"> • ((name bspfct1 bspfct2) param1 param2)) → Return aus Funktion name" wird danach mit param1 und param2 aufgerufen → Anwendung von bspfct1 auf param1 	4C/108
	filter	<ul style="list-style-type: none"> • Filterung abhängig von Prädikat • Beispiel: (define (my-filter prädikat liste)...) <ul style="list-style-type: none"> • (my-filter (lambda(x) (< x 10)) liste) • Durch Verwendung von Lambda → variable Filter 	4C/110
	map	<ul style="list-style-type: none"> • Anwendung einer Funktion auf jedes Element der Liste • Beispiel: (define (my-map fct list)...) <ul style="list-style-type: none"> • (my-map (lambda(x) (* x 10)) list) • Durch Verwendung von Lambda → variable Anpassung 	4C/125
	fold	<ul style="list-style-type: none"> • Zusammenrechnen einer Liste nach Vorschrift <ul style="list-style-type: none"> • foldr von rechts, foldl von links • Beispiel: (define (my-fold init fct list)...) <ul style="list-style-type: none"> • (my-fold 0 (lambda(x,y) (+ x y)) list) • Durch Verwendung von Lambda → variable Verrechnung 	4C/139

Listen	Syntax	• (define my-list (list 1 2 3))	4B/3
	cons	<ul style="list-style-type: none"> • Anhängen an Liste • z.B.: (define my-list2 (cons 0 list1)) → 0 1 2 3 	4B/6
	empty	• leere Liste (z.B.: (... (cons 0 empty)))	4B/8
	Funktionen	<ul style="list-style-type: none"> • (first my-list) → erstes Element • (rest my-list) → Alle Elemente außer Erstes 	4B/9
	Vertrag	<ul style="list-style-type: none"> • ;; Type: (list of ...) → ... • ;; Type: (list of ANY) → ... , falls Typ beliebig • ;; Type: X → X , Eingabetyp beliebig, Ausgabe vom selben Typ • ;; Type: ANY → ANY , Eingabetyp beliebig, Ausgabebetyp auch 	4B/16
	Rekursion	<ul style="list-style-type: none"> • Beispiel Summe: → (if (empty? list) 0 (+ (first list) (sum (rest list)))) • Beispiel Liste als Ausgabe: → .. (if (empty? list) empty (cons (sqrt (first list) (sqrts (rest list))))) 	4B/41
	Filter	<ul style="list-style-type: none"> • Aussortieren von gewissen Elementenn <ul style="list-style-type: none"> • (define (filter list) (if (Sortierbedingung) → (cons (first list) (filter list)) , falls #t → (filter list))) , falls #f 	4B/42
	homogen heterogen	<ul style="list-style-type: none"> • Listen auch mit unterschiedlichen Typen möglich → heterogen → ;; Type: (list of ANY)... 	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"> • grundlegendes Konzept zur Steuerung des Programmablaufs 	4A/161
-----------	-------	---	--------

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

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

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

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

		•	
		•	
		•	
		•	
		•	