

Pointfrip Quickinfo

2021-09-03

Im Folgenden geht es ums Programmieren auf Funktionsniveau mit Kombinatoren

Regel

In der Regel gilt **Rechts-vor-Links**, es gibt aber Ausnahmen z.B. bei der Kondition.

Für eine geänderte Auswertung der Terme müssen **Klammern** gesetzt werden.

Es gilt **Infixnotation** wie bei: $a + b$

Bei Funktionen schreibt man: $funktion \circ argument$

Datentypen

$[0]$, $[1]$, $[2]$, ... , $[i]$,
 $[_{123}]$ sind Selektoren, die auf die Werte einer Liste, einem Dict oder einem Array zugreifen; oder sind Integer*

name ist ein Bezeichner für eine ihm zugeordnete Funktion

$_{123.5678e_{30}}$ ist eine Real-Zahl

$(10 ; 20 ; 30 ; 40 ; 50 ;)$ ist eine Liste von Real-Zahlen

$(10 a 20 b 30 c 40 d 50 e)$ ist ein Dict* mit Werten und Schlüsseln

$()$ leere Liste

*man beachte, daß der Konstanten-Kombinator verwendet werden sollte.

Definiton von Funktionen/Konstanten/Operatoren

$bez == term$ weist dem Bezeichner einem Term zu

$cnst == 'literal$ Konstanten verwenden den Konstanten-Kombinator

$opr == (\dots) \circ ee$ Operatoren verwenden häufig ein ee und $[0]$ und $[1]$

Kombinatoren

<code>'name</code>	ist der Konstanten-Kombinator
<code>funktion1 ° funktion2</code>	ist die Komposition
<code>fun1 , fun2 , ... , funm ,</code>	ist die Konstruktion einer Liste
<code>(test -> dann sonst)</code>	ist der Kondition-Kombinator mit einem Alternat
<code>(test ->* term)</code>	ist eine While-Schleife
<code>(funktion aa)</code>	ist der Apply-To-All-Kombinator (map)
<code>(funktion \)</code>	ist der Insert-Kombinator (reduce)
<code>funktion1 ee funktion2</code>	wertet die Funktionen aus und erzeugt daraus ein Paar
<code>#name</code>	pickt den Wert zum Namen aus einem Dict

Funktionen

<code>id</code>	Identitätsfunktion
<code>iota</code>	erzeugt eine Liste von Zahlen ab 1 aufwärts bis zur Zahl
<code>head</code>	extrahiert den ersten Wert einer Liste
<code>tail</code>	extrahiert den Rest einer Liste
<code>infix</code>	extrahiert den Infixwert einer Liste/Dicts
<code>prop</code>	erzeugt eine Zelle aus Erstem,Infix,Rest,
<code>top</code>	wie head, aber nicht für Objekte
<code>pop</code>	wie tail, aber nicht für Objekte
<code>tag</code>	extrahiert den Typus oder Infixwert
<code>reverse</code>	kehrt eine Liste um

length	liefert die Länge einer Liste
sin	berechnet den Sinus einer Zahl
ln	berechnet den natürlichen Logarithmus einer Zahl
islist	prüft, ob es sich um eine Liste handelt

Operatoren

<i>erster , rest</i>	Komma erzeugt eine Liste
<i>num1 + num2</i>	Arithmetische Operatoren für Addition,
<i>num1 - num2</i>	für Subtraktion
<i>num1 * num2</i>	für Multiplikation
<i>num1 / num2</i>	für Division
<i>dict</i> get <i>bez</i>	ermittelt den Wert zum Schlüssel <i>bez</i> *
<i>dict</i> put <i>bez,wert,</i>	erzetzt/legt einen Schlüssel* mit Wert im Dict an
<i>(bez := wert) ° dict</i>	wie put, aber als Wert-Zuweisung im Dict
<i>num1 = num2</i>	prüft auf Gleichheit und liefert dann true oder false
etc	

Objekte und Klassen

<i>(turtle :: () stack 0 x 0 y ...)</i>	ist das Turtleobjekt mit den Attributen stack, x, y, ...
<i>turtle == .. { dict }</i>	ist die Turtleklasse mit den ... Methoden

Monaden und Effekte

<i>('turtle new) (draw eff 'io)</i>	erzeugt eine Monade zum Zeichnen der Turtlespur
-------------------------------------	---

io == .. { } sind die System-Effekte, sozusagen der Treiber (?)

et cetera zu finden in der Reference/[blaues Fragezeichen](#)

(CC0)

*man beachte, daß der Konstanten-Kombinator verwendet werden sollte.