

## Pointfrip Quickinfo

2023-07-07

Im Folgenden geht es ums Programmieren auf Funktionsniveau mit Kombinatoren

### Verarbeitungsregel

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 oder einem Dict zugreifen -- oder sind Integerzahlen*
name	ist ein Bezeichner für eine ihm zugeordnete Funktion
$_{123.5678e_{30}}$	ist eine Realzahl
$(10 ; 20 ; 30 ; 40 ; 50 ;)$	ist eine Liste von Realzahlen
$(10 a 20 b 30 c 40 d 50 e)$	ist ein Dict* mit Werten und Schlüsseln
$()$	leere Liste / <i>null</i>
$(head infix .. tail)$	Datenzelle / <i>prop</i>
"abcdef"	ist ein String
true / false	sind von Typ <i>bool</i>

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

## Definiton von Funktionen/Konstanten/Operatoren

<code>bez == term</code>	weist dem Bezeichner einem Term zu
<code>cnst == ' literal</code>	Konstanten verwenden den Konstanten-Kombinator
<code>opr == ( ... ) ° ee</code>	Operatoren verwenden häufig ein ee und [0] und [1]

## Kombinatoren

<code>'name</code>	ist der Konstanten-Kombinator
<code>funktion1 ° funktion2</code>	ist die Komposition, auch <code>o</code> verwendbar (right-pipe)
<code>fun1 , fun2 , ... , funm ,</code>	ist die Konstruktion einer Liste
<code>(test -&gt; dann ; sonst)</code>	ist der Kondition-Kombinator mit einer Alternative
<code>(test -&gt;* term)</code>	ist eine While-Schleife
<code>(funktion aa)</code>	ist der Apply-To-All-Kombinator (map)
<code>(funktion \ )</code>	ist der Insertr-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
<code>funktion : argument</code>	ist eine Appikation -- <code>funktion(argument)</code>
<code>liste insl 'func</code>	ist der Insertl-Operator; <code>insr</code> für Insertr-Operation
<code>func _s</code>	Single Funktion wird ausgeführt
<code>func1 app func2</code>	Apply-Funktion um Funktionale auszuführen
<code>func1 swee func2</code>	wie <code>ee</code> , nur die Elemente im Paar sind vertauscht
<code>(func aa0) ° liste,x,y, ... ,</code>	Mischung aus <code>aa</code> und <code>distr</code> , erweitert
<code>(liste,x,y, ... ,) map0 'func</code>	Mischung aus <code>map</code> und <code>distr</code> , erweitert
<code>liste filter 'boolfunc</code>	ist der Filter-Operator

## Listenverarbeitende Funktionen und Operatoren

<code>val0 ; val1 ; val2 ; ... ;</code>	Listenbildung mit literalen Werten
<code>head ° prop</code>	extrahiert den ersten Wert einer Liste
<code>tail ° prop</code>	extrahiert den Rest einer Liste
<code>infix ° prop</code>	extrahiert den Infixwert einer Liste/Dicts
<code>prop ° hd, inf, tl,</code>	erzeugt eine Datenzelle mit drei Werten
<code>term ° combi</code>	extrahiert den Term-Wert aus einem Combine-Datentyp
<code>arg ° combi</code>	extrahiert den Arg-Wert aus einem Combine-Datentyp
<code>type ° data</code>	liefert eine Bezeichnung für den Datentyp
<code>liste at num</code>	pickt den <i>num</i> -ten Wert aus der <i>liste</i>
<code>func , list</code>	das Komma fügt ein Element vor die Liste
<code>iota ° num</code>	erzeugt eine Liste von Zahlen ab 1 aufwärts bis <i>num</i>
<code>num1 to num2</code>	erzeugt eine Liste von Zahlen von <i>num1</i> bis <i>num2</i>
<code>reverse ° liste</code>	kehrt eine Liste um; funktioniert auch mit einem String
<code>trans ° matrix</code>	Transpose einer Liste von Listen (matrix)
<code>data distl liste</code>	Distribution Left
<code>liste distr data</code>	Distribution Right
<code>data make num</code>	erzeugt eine Liste mit <i>num</i> <i>data</i> -Werten
<code>liste take num</code>	liefert eine Liste der ersten <i>num</i> Elemente
<code>liste drop num</code>	liefert die Restliste ohne die ersten <i>num</i> Elemente
<code>liste1 ++ liste2</code>	liefert eine verkettete Liste
<code>length ° liste</code>	liefert die Länge einer Liste
<code>liste count data</code>	liefert die Anzahl von <i>data</i> in der Liste
<code>liste find data</code>	liefert die erste Position von <i>data</i> in der Liste

## Numerische Funktionen und Operatoren

$num1 + num2$	Addition von Zahlen gleichen Typs
$num1 - num2$	Subtraktion von Zahlen gleichen Typs
$num1 * num2$	Multiplikation von Zahlen gleichen Typs
$num1 / num2$	Division von Zahlen gleichen Typs
$num1 ^ num2$	Potenzierung von Zahlen gleichen Typs
$num1 \text{ idiv } num2$	Division von Integerzahlen
$num1 \text{ imod } num2$	Modulo von Integerzahlen
$\text{pred} \circ num$	Vorgängerfunktion
$\text{succ} \circ num$	Nachfolgerfunktion
$\text{sign} \circ num$	Signumfunktion
$\text{abs} \circ num$	Betrag der Zahl
$\text{neg} \circ num$	Negation der Zahl
$\_ \circ num$	Negation der Zahl
$\text{floor} \circ num$	Abrunden der Zahl
$\text{ceil} \circ num$	Aufrunden der Zahl
$\text{float} \circ num$	wandelt in eine Realzahl um
$\text{round} \circ num$	rundet in eine Integerzahl um
$\text{trunc} \circ num$	Integerzahl ohne Beachtung der Nachkommawerte
$\text{real roundto } num$	Rundet auf die $num$ -te Nachkommastelle

<b>exp</b> ° num	Exponentialfunktion der Zahl
<b>ln</b> ° num	natürlicher Logarithmus der Zahl
<b>lg</b> ° num	Zehnerlogarithmus der Zahl
<b>sq</b> ° num	das Quadrat der Zahl
<b>sqrt</b> ° num	die Quadratwurzel der Zahl
<b>cbrt</b> ° num	die Qubikwurzel der Zahl
<b>pi</b>	Funktion liefert die Zahl Pi
<b>2pi</b>	Funktion liefert den Umfang des Einheitskreises
<b>sin</b> ° num	Sinusfunktion der Zahl in Radiant
<b>cos</b> ° num	Cosinusfunktion der Zahl in Radiant
<b>tan</b> ° num	Tangensfunktion der Zahl in Radiant
<b>arcsin</b> ° num	Arcussinusfunktion
<b>arccos</b> ° num	Arcuscosinusfunktion
<b>arctan</b> ° num	Arcustangensfunktion
<b>y arctan2 x</b>	Phase (oder Arg) zu (x,y)
<b>sinh</b> ° num	Sinus Hyperbolicus Funktion
<b>cosh</b> ° num	Cosinus Hyperbolicus Funktion
<b>tanh</b> ° num	Tangens Hyperbolicus Funktion
<b>deg</b> ° num	wandelt Radiant in Degree um
<b>rad</b> ° num	wandelt Degree in Radiant um

## Boolische Funktionen und Operatoren

<i>data1 = data2</i>	prüft auf Gleichheit
<i>data1 != data2</i>	prüft auf Ungleichheit
<i>data1 &lt;&gt; data2</i>	prüft auf Ungleichheit, alternativ
<i>data1 &lt; data2</i>	Vergleich auf Kleiner-als
<i>data1 &gt; data2</i>	Vergleich auf Größer-als
<i>data1 &lt;= data2</i>	Vergleich auf Kleiner-Gleich
<i>data1 &gt;= data2</i>	Vergleich auf Größer-Gleich
<i>data1 min data2</i>	Minimum von <i>data1</i> und <i>data2</i>
<i>data1 max data2</i>	Maximum von <i>data1</i> und <i>data2</i>
<b>not</b> ° <i>bool</i>	Boolische Nicht-Funktion
<i>bool1 and bool2</i>	Boolische Und-Funktion
<i>bool1 or bool2</i>	Boolische Oder-Funktion
<i>bool1 xor bool2</i>	Boolische Exklusiv-Oder-Funktion
<b>isatom</b> ° <i>data</i>	Prüft, ob <i>data</i> zu den Atom-Typen gehört
<b>isnull</b> ° <i>data</i>	Prüft, ob <i>data</i> der Wert ( ) also Null ist
<b>isprop</b> ° <i>data</i>	Prüft, ob <i>data</i> eine Datenzelle ist
<b>islist</b> ° <i>data</i>	Prüft, ob <i>data</i> eine Liste ist
<b>isnum</b> ° <i>data</i>	Prüft, ob <i>data</i> eine Zahl ist, generisch
<b>iszero</b> ° <i>data</i>	Prüft, ob <i>data</i> die Zahl 0 ist, generisch
<b>ispos</b> ° <i>num</i>	Prüft, ob <i>num</i> eine positive Zahl ist, generisch
<b>isneg</b> ° <i>num</i>	Prüft, ob <i>num</i> eine negative Zahl ist, generisch
<b>isident</b> ° <i>data</i>	Prüft, ob <i>data</i> ein Bezeichner ist

<b>isint</b> ° <i>data</i>	Prüft, ob <i>data</i> eine Integerzahl ist
<b>isreal</b> ° <i>data</i>	Prüft, ob <i>data</i> eine Realzahl ist
<b>isstring</b> ° <i>data</i>	Prüft, ob <i>data</i> ein Zeichenstring ist
<b>iscons</b> ° <i>data</i>	Prüft, ob <i>data</i> eine List-Datenzelle ist
<b>isquote</b> ° <i>data</i>	Prüft, ob <i>data</i> ein Quotewert ist
<b>isivar</b> ° <i>data</i>	Prüft, ob <i>data</i> ein Instanz-Variablen-Selektor ist
<b>iscombi</b> ° <i>data</i>	Prüft, ob <i>data</i> ein Combine-Wert ist
<b>isact</b> ° <i>data</i>	Prüft, ob <i>data</i> ein Act-Wert ist
<b>isbool</b> ° <i>data</i>	Prüft, ob <i>data</i> ein boolischer Wert ist
<b>isbound</b> ° <i>ident</i>	Prüft, ob der Bezeichner schon definiert ist
<b>isundef</b> ° <i>data</i>	Prüft, ob <i>data</i> der Wert <b>_undef</b> ist
<i>data</i> <b>in</b> <i>liste</i>	Prüft, ob <i>data</i> als Element in der <i>liste</i> enthalten ist

## Dict Funktionen und Operatoren

<b>#ident</b> ° <i>dict</i>	der Selektor pickt zum <i>ident</i> -Key den Wert aus dem <i>dict</i>
<i>dict</i> <b>iget</b> <i>ident</i>	zum <i>ident</i> *-Key wird der Wert aus dem <i>dict</i> herausgepickt
<i>dict</i> <b>iput</b> <i>ident</i> , <i>value</i> ,	zum <i>ident</i> *-Key wird der <i>value</i> neu im <i>dict</i> angelegt
<i>dict</i> <b>get</b> <i>key</i>	zum <i>key</i> wird der Wert aus dem <i>dict</i> herausgepickt
<i>dict</i> <b>put</b> <i>key</i> , <i>value</i> ,	zum <i>key</i> wird der <i>value</i> neu im <i>dict</i> angelegt
( <i>ident</i> <b>:=</b> <i>func</i> ) ° <i>dict</i>	wie bei <b>iput</b> geschieht diese "Variablen"-Zuweisung
( <i>func</i> <b>&lt;-</b> <i>x</i> ; <i>y</i> ; ... ;) ° <i>liste</i>	<i>func</i> wendet das erzeugte Dict, wie nach einem Assign, an
<b>keys</b> ° <i>dict</i>	erzeugt eine Liste mit allen Keys aus dem <i>dict</i>
<b>values</b> ° <i>dict</i>	erzeugt eine Liste mit allen Values aus dem <i>dict</i>
<b>it</b> ° <i>dict</i>	Pickt aus dem <i>dict</i> den zu <b>_it</b> zugehörigen Wert

## String Funktionen und Operatoren

<b>length</b> ° <i>string</i>	gibt die Länge des Strings an
<b>substring</b> ° <i>string, i, len,</i> <i>string1 &amp; string2</i>	Kopiert einen Teilstring aus <i>string</i> verkettet zwei Strings
<i>string1</i> <b>concat</b> <i>string2</i>	verkettet zwei Strings
<i>string</i> <b>indexOf</b> <i>substr</i>	sucht die Position von <i>substr</i> im <i>string</i> von links
<b>trim</b> ° <i>string</i>	schneidet links und rechts die Leerzeichen ab
<b>triml</b> ° <i>string</i>	schneidet links die Leerzeichen ab
<b>trimr</b> ° <i>string</i>	schneidet rechts die Leerzeichen ab
<b>upper</b> ° <i>string</i>	wandelt den String in Großbuchstaben um
<b>lower</b> ° <i>string</i>	wandelt den String in Kleinbuchstaben um
<b>capitalize</b> ° <i>string</i>	wandelt den String in ein Hauptwort um
<b>char</b> ° <i>num</i>	erzeugt ein Zeichen nach dem Unicodewert
<b>unicode</b> ° <i>string</i>	bestimmt den Unicodewert des ersten Zeichens
<b>parse</b> ° <i>string</i>	parst den String mit dem Pointfrip-Parser
<b>value</b> ° <i>string</i>	wandelt Zahlen, Wörter, Listen im String in Daten um
<b>string</b> ° <i>data</i>	wandelt die Daten in einen Printstring um
<b>unpack</b> ° <i>string</i>	zerlegt den String in eine Liste von Einzelzeichen
<i>string</i> <b>split</b> <i>delstr</i>	zerlegt den <i>string</i> in eine Liste mit Strings ohne <i>delstr</i>
<i>liste</i> <b>join</b> <i>insstr</i>	verbindet die Strings der Liste mit <i>insstr</i> dazwischen

## Matrix Funktionen und Operatoren

<i>matrix1</i> <b>add</b> <i>matrix2</i>	Addiert zwei Matrizen, komponentenweise
<i>matrix1</i> <b>sub</b> <i>matrix2</i>	Subtrahiert <i>matrix2</i> von <i>matrix1</i>
<i>matrix1</i> <b>mul</b> <i>matrix2</i>	Multipliziert zwei Matrizen
<i>num</i> <b>mul</b> <i>matrix</i> <i>matrix</i> <b>mul</b> <i>num</i>	Multipliziert die <i>matrix</i> mit einem Skalarwert
<b>ismat</b> ° <i>data</i>	Prüft, ob <i>data</i> eine Matrix ist, vereinfachte Form
<b>trans</b> ° <i>matrix</i>	Transpose der <i>matrix</i>
<b>det</b> ° <i>matrix</i>	Berechnet die Determinante der <i>matrix</i>
<b>inv</b> ° <i>matrix</i>	Berechnet die Inverse Matrix
<i>num1</i> <b>zeromat</b> <i>num2</i>	erzeugt eine Matrix mit lauter Nullen
<b>idmat</b> ° <i>num</i>	Identitätsmatrix der Größe <i>num</i>
<b>fail</b> ° <i>infodata</i>	erzeugt Standardfehlermeldung für einen Fail
<i>liste</i> <b>IP</b> <i>liste</i>	Inner Product nach John Backus
<i>matrix</i> <b>MM</b> <i>matrix</i>	Matrixmultiplikation nach John Backus
<b>rnd</b> ° <i>matrix</i>	Rundet <i>matrix</i> auf fünf Nachkommastellen
<b>zero</b> ° <i>data</i> <b>zero</b> ° <i>matrix</i>	generiert eine Null, typenabhängig
<b>one</b> ° <i>data</i> <b>one</b> ° <i>matrix</i>	generiert eine Eins, typenabhängig

## Misc Funktionen und Operatoren

<b>undef</b>	erzeugt Fehlermeldung für undefinierte Funktion
<b>id</b> ° <i>data</i>	Identitätsfunktion liefert <i>data</i>
<b>name</b> ° <i>ident</i>	extrahiert den String des Bezeichners
<b>body</b> ° <i>ident</i>	extrahiert den Definitionswert des Bezeichners
<b>info</b> ° <i>ident</i>	extrahiert den Compilerstring des Bezeichners
<b>identlist</b>	gibt eine Liste mit allen verwendeten Bezeichnern aus
<b>quote</b> ° <i>data</i>	macht aus <i>data</i> einen Quote-Wert
<i>ident</i> <b>error</b> <i>string</i>	gibt eine Fehlermeldung mit <i>ident</i> und <i>string</i> aus
' <i>func1</i> <b>comp</b> ' <i>func2</i>	verkettet die Funktionen zu einer neuen Funktion
<i>int</i> <b>act</b> <i>dict</i>	erzeugt einen Act-Wert mit den Daten - (Monade)
<i>act</i> <b>bind</b> ' <i>func</i>	legt die <i>func</i> im bind-Feld von einem neuen <i>act</i> an
<i>act</i> >> <i>func</i>	legt die <i>func</i> im bind-Feld von <i>einem neuen act</i> an
<i>fname</i> <b>load</b>	liest den Text von der Datei <i>fname</i> ins Display
<i>fname</i> <b>save</b>	sichert den Text vom Display in die Datei <i>fname</i>
<b>files</b>	gibt eine Liste mit allen Dateinamen aus
<i>fname</i> <b>loadtext</b>	lädt den String aus der Datei <i>fname</i> im Ordner "pf/"
<i>fname</i> <b>savetext</b> <i>string</i>	sichert den <i>string</i> in der Datei <i>fname</i> im Ordner "pf/"
<b>stopvm</b>	bricht die Berechnung ab, mit Fehlermeldung
<b>dump</b>	zeigt alle Bezeichner mit ihren Zuordnungen an
<b>savedump</b> (for test)	zeigt alle info-Strings der Bezeichner an
<b>help</b>	Linkadresse zu aktuellem Hilfe-PDF
<b>pim</b> ° <i>num</i>	gibt eine Liste aller Primfaktoren einer Zahl an, Beispiel
( <i>test</i> <b>try then;else</b> )° <i>argum</i>	Prüft <i>test</i> auf Error -> then/else mit ( <i>result</i> ; <i>argum</i> ;)

## Hinweise zum Laden und Speichern von Programmdateien

<code>"filename" save</code>	ein Programmtext wird unter dem Namen <i>filename</i> im Ordner „pf/“ gesichert
<code>"filename" load</code>	ein Programmtext der Datei <i>filename</i> aus dem Ordner „pf/“ wird mit den Definitionen eingelesen
<code>files</code>	gibt Liste mit allen Dateinamen im Ordner „pf/“ aus

Mit **identlist** oder **dump** bekommt man einen Überblick über die verwendeten Wörter.

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

(CC0)