Pointfrip Quickinfo

2023-06-19

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* **°** *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 / *null*
(*head infix* **..** *tail*)	Datenzelle / *prop*
"abcdef"	ist ein String
true / false	sind von Typ *bool*

*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 **==** (...) **°** **ee**	Operatoren verwenden häufig ein ee und [0] und [1]

Kombinatoren

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

Listenverarbeitende Funktionen und Operatoren

val0 **;** *val1* **;** *val2* **;** ... **;**	Listenbildung mit literalen Werten
head ° *prop*	extrahiert den ersten Wert einer Liste
tail ° *prop*	extrahiert den Rest einer Liste
infix ° *prop*	extrahiert den Infixwert einer Liste/Dicts
prop ° *hd*, *inf*, *tl*,	erzeugt eine Datenzelle mit drei Werten
term ° *combi*	extrahiert den Term-Wert aus einem Combine-Datentyp
arg ° *combi*	extrahiert den Arg-Wert aus einem Combine-Datentyp
type ° *data*	liefert eine Bezeichnung für den Datentyp
func **,** *list*	das Komma fügt ein Element vor die Liste
iota ° *num*	erzeugt eine Liste von Zahlen ab 1 aufwärts bis *num*
num1 **to** *num2*	erzeugt eine Liste von Zahlen von *num1* bis *num2*
reverse ° *liste*	kehrt eine Liste um; funktioniert auch mit einem String
trans ° *matrix*	Transpose einer Liste von Listen (matrix)
data **distl** *liste*	Distribution Left
liste **distr** *data*	Distribution Right
data **make** *num*	erzeugt eine Liste mit *num data*-Werten
liste **take** *num*	liefert eine Liste der ersten *num* Elemente
liste **drop** *num*	liefert die Restliste ohne die ersten *num* Elemente
liste1 **++** *liste2*	liefert eine verkettete Liste
length ° *liste*	liefert die Länge einer Liste
liste **count** *data*	liefert die Anzahl von *data* in der Liste
liste **find** *data*	liefert die erste Position von *data* 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 **idiv** *num2*	Division von Integerzahlen
num1 **imod** *num2*	Modulo von Integerzahlen
pred ° *num*	Vorgängerfunktion
succ ° *num*	Nachfolgerfunktion
sign ° *num*	Signumfunktion
abs ° *num*	Betrag der Zahl
neg ° *num*	Negation der Zahl
_ ° *num*	Negation der Zahl
floor ° *num*	Abrunden der Zahl
ceil ° *num*	Aufrunden der Zahl
float ° *num*	wandelt in eine Realzahl um
round ° *num*	rundet in eine Integerzahl um
trunc ° *num*	Integerzahl ohne Beachtung der Nachkommawerte
real **roundto** *num*	Rundet auf die *num*-te Nachkommastelle

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

Boolische Funktionen und Operatoren

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

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

Dict Funktionen und Operatoren

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

String Funktionen und Operatoren

length ° *string*	gibt die Länge des Strings an
substring ° *string*, *i*, *len*,	Kopiert einen Teilstring aus *string*
string1 **&** *string2*	verkettet zwei Strings
string1 **concat** *string2*	verkettet zwei Strings
string **indexof** *substr*	sucht die Position von *substr* im *string* von links
trim ° *string*	schneidet links und rechts die Leerzeichen ab
triml ° *string*	schneidet links die Leerzeichen ab
trimr ° *string*	schneidet rechts die Leerzeichen ab
upper ° *string*	wandelt den String in Großbuchstaben um
lower ° *string*	wandelt den String in Kleinbuchstaben um
capitalize ° *string*	wandelt den String in ein Hauptwort um

char ° *num*	erzeugt ein Zeichen nach dem Unicodewert
unicode ° *string*	bestimmt den Unicodewert des ersten Zeichens
parse ° *string*	parst den String mit dem Pointfrip-Parser
value ° *string*	wandelt Zahlen, Wörter,Listen im String in Daten um
string ° *data*	wandelt die Daten in einen Printstring um
unpack ° *string*	zerlegt den String in eine Liste von Einzelzeichen
string **split** *delstr*	zerlegt den *string* in eine Liste mit Strings ohne *delstr*
liste **join** *insstr*	verbindet die Strings der Liste mit *insstr* dazwischen

Matrix Funktionen und Operatoren

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

Misc Funktionen und Operatoren

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

Hinweise zum Laden und Speichern von Programmdateien

"*filename*" **save**	ein Programmtext wird unter dem Namen *filename* im Ordner „pf/“ gesichert
"*filename*" **load**	ein Programmtext der Datei *filename* aus dem Ordner „pf/“ wird mit den Definitionen eingelesen
files	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.