

Pointfrip Quickinfo

2023-11-25

The following is about programming at the **function-level** with combinators

--- bitte Calibri und mich dann löschen ---

Rule

As a rule, **right-before-left** applies, but there are exceptions, e.g. in condition terms.

Parentheses must be used to change the evaluation of the terms.

Infix notation applies as in: $a + b$

For functions you write: $function \circ argument$

Data Types

$[0], [1], [2], \dots, [i],$
 $[_{123}]$ are selectors that access the values of a list
or a dict -- or are integer* numbers

name is an identifier for an associated function

$_{123.5678e_{30}}$ is a real number

$(10 ; 20 ; 30 ; 40 ; 50 ;)$ is a list of real numbers

$(10\ a\ 20\ b\ 30\ c\ 40\ d\ 50\ e)$ is a dict* with values and keys

$()$ empty list / *null*

$(head\ infix\ ..\ tail)$ data cell / *prop*

"abcdef" is a string

true / false are of type *bool*

*note that the constant combinator should be used.

Definition of Functions/Constants/Operators

<i>identifier == term</i>	assigns a term to the identifier
<i>constname == ' literal</i>	Constants use the constant combinator
<i>oprname == (...) ° ee</i>	Operators often use an ee and [0] and [1]

Combinators

<i>'name</i>	is the Constant combinator	
<i>function1 ° function2</i>	is the Composition, o can also be used	(right-pipe)
<i>fun1 , fun2 , ... , funm ,</i>	is the Construction of a list	
<i>(test -> then ; else)</i>	is the Condition combinator with an alternative	
<i>(test -> * term)</i>	is a While loop	
<i>(function aa)</i>	is the Apply-to-All combinator	(map)
<i>(function \)</i>	is the Inserttr combinator	(reduce)
<i>function1 ee function2</i>	evaluates the functions and creates a pair from them	
<i>#name</i>	picks the value for the name from a dict	
<i>function : argument</i>	is an Application -- <i>function(argument)</i>	
<i>list insl 'func</i>	is the Insertl operator; insr for Inserttr operation	
<i>func _s</i>	Single function is executed	
<i>func1 app func2</i>	Apply function to execute Functionals	
<i>func1 swee func2</i>	like ee , only the elements in the pair are swapped	
<i>(func aa0) ° list,x,y, ... ,</i>	Mixture of aa and distr , expanded	
<i>(list,x,y, ... ,) map0 'func</i>	Mixture of map and distr , extended	
<i>list filter 'boolfunc</i>	is the Filter operator	

List Processing Functions and Operators

<code>val0 ; val1 ; val2 ; ... ;</code>	building lists with literal values
<code>head ° prop</code>	extracts the first value of a list
<code>tail ° prop</code>	extracts the rest of a list
<code>infix ° prop</code>	extracts the infix value of a list/dict
<code>prop ° hd, inf, tl,</code>	creates a data cell with three values
<code>term ° combi</code>	extracts the Term value from a Combine data type
<code>arg ° combi</code>	extracts the Arg value from a Combine data type
<code>type ° data</code>	supplies a name for the data type
<code>list at num</code>	picks the <i>num</i> -th value from the <i>list</i>
<code>func , list</code>	the comma adds an element before the <i>list</i>
<code>iota ° num</code>	creates a list of numbers from 1 upwards to <i>num</i>
<code>num1 to num2</code>	produces a list of numbers from <i>num1</i> to <i>num2</i>
<code>reverse ° list</code>	reverses a list; also works with a string
<code>trans ° matrix</code>	Transpose a list of lists (matrix)
<code>data distl list</code>	Distribution Left
<code>list distr data</code>	Distribution Right
<code>data make num</code>	creates a list of <i>num</i> <i>data</i> -values
<code>list take num</code>	returns a list of the first <i>num</i> elements
<code>list drop num</code>	returns the remainder list without the first <i>num</i> elements
<code>list1 ++ list2</code>	concatenates two lists into a new list
<code>length ° list</code>	returns the length of a <i>list</i>
<code>list count data</code>	returns the number of <i>data</i> in the <i>list</i>
<code>list find data</code>	returns the first position of <i>data</i> in the <i>list</i>

Numerical Functions and Operators

$num1 + num2$	Addition of numbers of the same type
$num1 - num2$	Subtraction of numbers of the same type
$num1 * num2$	Multiplication of numbers of the same type
$num1 / num2$	Division of numbers
$num1 ^ num2$	Exponentiation of numbers of the same type
$num1 \text{ idiv } num2$	Division of integer numbers
$num1 \text{ imod } num2$	Modulo of integer numbers
$\text{pred} \circ num$	Predecessor function
$\text{succ} \circ num$	Successor function
$\text{sign} \circ num$	Signum function
$\text{abs} \circ num$	Absolute function
$\text{neg} \circ num$	Negation of the number
$_ \circ num$	Negation of the number
$\text{floor} \circ num$	Rounding down the number
$\text{ceil} \circ num$	Round up the number
$\text{float} \circ num$	converts to a real number
$\text{round} \circ num$	rounds to an integer number
$\text{trunc} \circ num$	Integer number with truncation of the decimal places
$\text{real roundto } num$	rounds to the num -th decimal place

exp ° num	Exponential function of the number
ln ° num	Natural logarithm of the number
lg ° num	Ten logarithm of the number
sq ° num	the square of the number
sqrt ° num	the square root of the number
cbrt ° num	the cube root of the number
pi	Function returns the number Pi
2pi	Function returns the perimeter of the unit circle
sin ° num	Sine function of the number in radians
cos ° num	Cosine function of the number in radians
tan ° num	Tangent function of the number in radians
arcsin ° num	Arcsine function
arccos ° num	Arccosine function
arctan ° num	Arc tangent function
y arctan2 x	Phase (or Arg) to (x,y)
sinh ° num	Hyperbolic sine function
cosh ° num	Hyperbolic cosine function
tanh ° num	Hyperbolic tangent function
deg ° num	converts radians to degrees
rad ° num	converts degree to radian

Boolean Functions and Operators

<i>data1</i> = <i>data2</i>	checks for equality
<i>data1</i> != <i>data2</i>	checks for inequality
<i>data1</i> <> <i>data2</i>	checks for inequality, alternatively
<i>data1</i> < <i>data2</i>	Compare to less than
<i>data1</i> > <i>data2</i>	Compare to greater-than
<i>data1</i> <= <i>data2</i>	Comparison on less than or equal
<i>data1</i> >= <i>data2</i>	Greater-equal comparison
<i>data1</i> min <i>data2</i>	Minimum of <i>data1</i> and <i>data2</i>
<i>data1</i> max <i>data2</i>	Maximum of <i>data1</i> and <i>data2</i>
not ° <i>bool</i>	Boolean Not-function
<i>bool1</i> and <i>bool2</i>	Boolean And function
<i>bool1</i> or <i>bool2</i>	Boolean OR function
<i>bool1</i> xor <i>bool2</i>	Boolean Exclusive-Or function
isatom ° <i>data</i>	Checks whether <i>data</i> belongs to the Atom types
isnull ° <i>data</i>	Checks whether <i>data</i> is the value (), i.e. null
isprop ° <i>data</i>	Checks whether <i>data</i> is a data cell / <i>prop</i>
islist ° <i>data</i>	Checks whether <i>data</i> is a list
isnum ° <i>data</i>	Tests whether <i>data</i> is a number, generic
iszero ° <i>data</i>	Tests whether <i>data</i> is the number 0, generic
ispos ° <i>num</i>	Tests whether <i>num</i> is a positive number, generic
isneg ° <i>num</i>	Tests whether <i>num</i> is a negative number, generic
isident ° <i>data</i>	Checks whether <i>data</i> is an identifier

isint ° <i>data</i>	Tests whether <i>data</i> is an integer number
isreal ° <i>data</i>	Checks whether <i>data</i> is a real number
isstring ° <i>data</i>	Checks whether <i>data</i> is a character string
iscons ° <i>data</i>	Checks whether <i>data</i> is a List data cell
isquote ° <i>data</i>	Checks whether <i>data</i> is a Quote value
isivar ° <i>data</i>	Checks whether <i>data</i> is an instance variable selector
iscombi ° <i>data</i>	Checks whether <i>data</i> is a Combine value
isact ° <i>data</i>	Checks whether <i>data</i> is an Act value
isbool ° <i>data</i>	Checks whether <i>data</i> is a boolean value
isbound ° <i>ident</i>	Checks whether the identifier is already defined
isundef ° <i>data</i>	Checks whether <i>data</i> is the value _undef
<i>data</i> in <i>list</i>	Checks whether <i>data</i> is included as an element in the <i>list</i>

Dict Functions and Operators

#ident ° <i>dict</i>	the selector picks the value from the <i>dict</i> for the <i>ident</i> key
<i>dict</i> iget <i>ident</i>	for the <i>ident</i> * key, the value is picked out of the <i>dict</i>
<i>dict</i> iput <i>ident</i> , <i>value</i> ,	the <i>value</i> for the <i>ident</i> * key is newly created in the <i>dict</i>
<i>dict</i> get <i>key</i>	for the <i>key</i> , the value is picked out of the <i>dict</i>
<i>dict</i> put <i>key</i> , <i>value</i> ,	the <i>value</i> for the <i>key</i> is newly created in the <i>dict</i>
(<i>ident</i> := <i>func</i>) ° <i>dict</i>	as with iput , this "variable" assignment occurs
(<i>func</i> <- <i>x</i> ; <i>y</i> ; ... ;) ° <i>list</i>	<i>func</i> applies the generated Dict, as after an assign
keys ° <i>dict</i>	creates a list with all Keys from the <i>dict</i>
values ° <i>dict</i>	creates a list with all Values from the <i>dict</i>
it ° <i>dict</i>	picks the value associated with _it from the <i>dict</i>

String Functions and Operators

length ° <i>string</i>	specifies the length of the string
substring ° <i>string</i> , <i>i</i> , <i>len</i> ,	copies a substring from <i>string</i>
<i>string1</i> & <i>string2</i>	concatenates two strings
<i>string1</i> concat <i>string2</i>	concatenates two strings
<i>string</i> indexof <i>substr</i>	searches the position of <i>substr</i> in the <i>string</i> from the left
trim ° <i>string</i>	cuts off the spaces left and right
triml ° <i>string</i>	cuts off the spaces on the left
trimr ° <i>string</i>	cuts off the spaces on the right
upper ° <i>string</i>	converts the string to uppercase
lower ° <i>string</i>	converts the string to lowercase
capitalize ° <i>string</i>	converts the string into a capital word
char ° <i>num</i>	produces a character according to the Unicode value
unicode ° <i>string</i>	specifies the Unicode value of the first character
parse ° <i>string</i>	parses the string with the Pointfrip-parser
value ° <i>string</i>	converts numbers, words, lists in the <i>string</i> into data
string ° <i>data</i>	converts the <i>data</i> into a print string
unpack ° <i>string</i>	breaks the <i>string</i> into a list of individual characters
<i>string</i> split <i>delstr</i>	breaks the <i>string</i> into a list of strings without <i>delstr</i>
<i>list</i> join <i>insstr</i>	connects the strings of the <i>list</i> with <i>insstr</i> in between

Matrix Functions and Operators

<i>matrix1</i> add <i>matrix2</i>	Adds two matrices, component by component
<i>matrix1</i> sub <i>matrix2</i>	Subtracts <i>matrix2</i> from <i>matrix1</i>
<i>matrix1</i> mul <i>matrix2</i>	Multiplies two matrices
<i>num</i> mul <i>matrix</i> <i>matrix</i> mul <i>num</i>	Multiplies the matrix by a scalar value
ismat ° <i>data</i>	Checks whether <i>data</i> is a matrix, simplified form
trans ° <i>matrix</i>	Transpose the <i>matrix</i>
det ° <i>matrix</i>	calculates the Determinant of the <i>matrix</i>
inv ° <i>matrix</i>	calculates the Inverse matrix
<i>num1</i> zeromat <i>num2</i>	creates a matrix with all zeros
idmat ° <i>num</i>	Identity matrix of size <i>num</i>
fail ° <i>infodata</i>	generates standard error message for a fail
<i>list</i> IP <i>list</i>	Inner Product according to John Backus
<i>matrix</i> MM <i>matrix</i>	Matrix multiplication according to John Backus
rnd ° <i>matrix</i>	Rounds <i>matrix</i> to five decimal places
zero ° <i>data</i> zero ° <i>matrix</i>	generates a Zero, depending on the type
one ° <i>data</i> one ° <i>matrix</i>	generates a One, depending on the type

Misc Functions and Operators

undef	generates error message for undefined function
id ° <i>data</i>	Identity function returns <i>data</i>
name ° <i>ident</i>	extracts the string of the identifier
body ° <i>ident</i>	extracts the definition value of the identifier
info ° <i>ident</i>	extracts the compiler-string of the identifier
identlist	outputs a list of all used identifiers
quote ° <i>data</i>	turns <i>data</i> into a Quote value
<i>ident</i> error <i>string</i>	outputs an error message with <i>ident</i> and <i>string</i>
' <i>func1</i> comp ' <i>func2</i>	chains the functions into a new function
<i>int</i> act <i>dict</i>	creates an Act value with the data - (Monade)
<i>act</i> bind ' <i>func</i>	creates the <i>func</i> in the bind field of a new <i>act</i>
<i>act</i> >> <i>func</i>	creates the <i>func</i> in the bind field of a new <i>act</i>
<i>fname</i> load	reads the text from the file <i>fname</i> into the display
<i>fname</i> save	saves the text from the display to the file <i>fname</i>
files	outputs a list with all file names
<i>fname</i> loadtext	loads the string from the file <i>fname</i> in the "pf/" folder
<i>fname</i> savetext <i>string</i>	saves the <i>string</i> in the file <i>fname</i> in the "pf/" folder
stopvm	the calculation aborts with an error message
dump	displays all identifiers with their assignments
savedump (for test)	displays all info-strings of the identifiers
help	Link address to current help-PDF
pim ° <i>num</i>	gives a list of all prime factors of a number, example
(<i>test</i> try <i>then</i> ; <i>else</i>) ° <i>argum</i>	Checks <i>test</i> for Error -> <i>then/else</i> with (<i>result</i> ; <i>argum</i> ;)

Notes on Loading and Saving Program Files

"filename" save	a program text is saved under the name <i>filename</i> in the "pf/" folder
"filename" load	a program text from the file <i>filename</i> from the "pf/" folder is read in with the definitions
files	outputs a list of all file names in the "pf/" folder

With **identlist** or **dump** you get an overview of the used words.

Expansion of Prelude with some Definitions

(num r) ° list	accesses the <i>num</i> -th element from the back of the <i>list</i>
tailr ° list	copy of the <i>list</i> without the last element
last ° list	the last element of the <i>list</i>
rotl ° list	rotation of the list elements to the left direction
rotr ° list	rotation of the list elements to the right direction
'expr times num,initakku,	repeats <i>expr</i> <i>num</i> times with <i>initakku</i> as start argument
foldl ° 'expr,initakku,list,	reduces the <i>list</i> with <i>expr</i> from the left side with <i>initakku</i> as the starting value
foldr ° 'expr,initakku,list,	reduces the <i>list</i> with <i>expr</i> from the right side with <i>initakku</i> as the starting value

*note that the constant combinator should be used.

(CC0)