

# Pointfrip Language Reference

2022-07-14

## Naming Conventions

<b>name</b>	the name itself
<i>name</i>	the type / the class
(?)	Uncertainties
*	Footnote / Note

## Data Types

<u>Data type</u>	<u>Syntax</u>	<u>Type identifier</u>
<i>data</i>		// General
<i>null</i>	( )	_null
<i>int*</i>	[_123]	_integer
<i>real</i>	_31.415e_123	_real
<i>string</i>	"abc"	_string
<i>ident</i>	abc <u>or</u> +-*/	_ident
<i>prefix</i>	@...	_prefix
<i>index</i>	[abc]	_index
<i>array</i>	{a b c}	_array
<i>error</i>	( <i>index</i> _error ... ..)	_error
<i>table*/dict*</i>	(a x b y c z)	// In pairs
<i>list</i>	(a ; b ; c ;)	;
<i>object</i>	( <i>ident</i> :: a x b y c z)	::
<i>turtle</i>	(turtle :: ... ..)	// Object
<i>combi</i>	( <i>term</i> _combine .. <i>arg</i> )	_combine
<i>monad</i>	( <i>int</i> _act ... ..)	_act
<i>bool</i>	true <u>or</u> false	// Idents
etc...		

\* note that *int*- and *dict*-literals require the constant combinator!

## Comments

*codetext* // comment

## Definition of Identifiers

*ident* == *term*

*ident* ≡ *term*

## Script Structure

*term* *definition1* *definition2* *definition3* ...

## Ddot

*prop* = *head* *infix* .. *tail*

## Include Files

**coreimport** == "Script1.txt" ; "Script2.txt" ; "Script3.txt" ; ... ;

**userimport** == "Script1.txt" ; "Script2.txt" ; "Script3.txt" ; ... ;

corepath?

userpath?

## List/Dict Functions and Operators

$dict = (first_1 \text{ infix}_1 first_2 \text{ infix}_2 \dots first_m \text{ infix}_m)$

$list = (element_0 ; element_1 ; element_2 ; \dots ; )$

$[i] \circ list \quad \quad \quad -- \quad \quad element_i$

$head \circ dict \quad \quad \quad -- \quad \quad first$

$head \circ list \quad \quad \quad -- \quad \quad first$

First element of the list.

$head \circ object \quad \quad \quad --$

$tail \circ dict \quad \quad \quad -- \quad \quad rest$

$tail \circ list \quad \quad \quad -- \quad \quad rest$

List without the first element and first infix.

$tail \circ object \quad \quad \quad --$

$infix \circ dict \quad \quad \quad -- \quad \quad infix \text{ value}$

$infix \circ object \quad \quad \quad --$

$prop \circ first, infix, rest, \quad \quad \quad -- \quad \quad dict$

$top \circ dict \quad \quad \quad -- \quad \quad first$

$top \circ list \quad \quad \quad -- \quad \quad first$

$pop \circ dict \quad \quad \quad -- \quad \quad rest$

$pop \circ list \quad \quad \quad -- \quad \quad rest$

$tag \circ data \quad \quad \quad -- \quad \quad typus \quad \quad \quad // \text{ als typeof}$

$tag \circ dict \quad \quad \quad -- \quad \quad infix \text{ value}$

$term \circ combi \quad \quad \quad -- \quad \quad term \text{ value}$

$arg \circ combi \quad \quad \quad -- \quad \quad arg \text{ value}$

$termoarg \quad \quad \quad -- \quad \quad term \text{ o } arg$

$first, rest \quad \quad \quad -- \quad \quad list$

Appendleft

$length \circ dict \quad \quad \quad -- \quad \quad real$

$length \circ list \quad \quad \quad -- \quad \quad real$

$length \circ object \quad \quad \quad --$

Number of list items.

$reverse \circ dict \quad \quad \quad -- \quad \quad dict$

$reverse \circ list \quad \quad \quad -- \quad \quad list$

$reverse \circ object \quad \quad \quad --$

Reverses the list items.

*data* **distl** *list*            --        *matrix*

*list* **distr** *data*            --        *matrix*

*dict* ++ *dict*                --        *dict*

*list* ++ *list*                --        *list*

Concatenate the lists.

*dict* **take** *num*              --        *dict*

*list* **take** *num*              --        *list*

Takes the first *num* elements from the list.

*dict* **drop** *num*              --        *dict*

*list* **drop** *num*              --        *list*

Drops the first *num* elements in the list.

**trans** ° *matrix*              --        *matrix*

**transpose** ° *matrix*        --        *matrix*

*num* **pick** *list*              --        *element*

*num* **sel** *list*                --        *element*

**last** ° *list*                 --

(*num* **r**) ° *list*              --

**tailr** ° *list*                --        *list*

**tailr** ° *dict*                --        *dict*

**rotl** ° *list*                 --        *list*

**rotr** ° *list*                 --        *list*

*list* **count** *data*            --        *real*

*data* **make** *num*              --        *list*

*list* **find** *data*            --        *real*

**iota** ° *num*                 --        *list*

**ι** ° *num*                    --        *list*

Generates a list of numbers from 1 to *num*.

**iota0** ° *num*                --        *list*

Generates a list of numbers from 0 to *num*-1.

*int* **to** *int*                 --        *list*

*real* **to** *real*                --        *list*

*int* **upto** *int*                --        *list*

*real* **upto** *real*            --        *list*

<i>int</i> <b>downto</b> <i>int</i>	--	list
<i>real</i> <b>downto</b> <i>real</i>	--	list
<b>swap</b> ° <i>x,y,list</i>	--	<i>y,x,list</i>

## Math Functions and Operators

<i>int</i> + <i>int</i>	--	<i>int</i>
<i>real</i> + <i>real</i>	--	<i>real</i>

Addition of numbers.

<i>int</i> - <i>int</i>	--	<i>int</i>
<i>real</i> - <i>real</i>	--	<i>real</i>

Subtraction of numbers.

<i>int</i> * <i>int</i>	--	<i>int</i>
<i>real</i> * <i>real</i>	--	<i>real</i>
<i>int</i> × <i>int</i>	--	<i>int</i>
<i>real</i> × <i>real</i>	--	<i>real</i>

Multiplication of numbers.

<i>num</i> / <i>num</i>	--	<i>real</i>
<i>num</i> ÷ <i>num</i>	--	<i>real</i>

Division of numbers.

<i>int</i> ^ <i>int</i>	--	<i>int</i>
<i>real</i> ^ <i>real</i>	--	<i>real</i>

Power of numbers.

<i>int</i> <b>idiv</b> <i>int</i>	--	<i>int</i>
-----------------------------------	----	------------

Integer division

<i>int</i> <b>imod</b> <i>int</i>	--	<i>int</i>
-----------------------------------	----	------------

Integer modulo

<b>pred</b> ° <i>int</i>	--	<i>int</i>
<b>pred</b> ° <i>real</i>	--	<i>real</i>

Predecessor function

<b>succ</b> ° <i>int</i>	--	<i>int</i>
<b>succ</b> ° <i>real</i>	--	<i>real</i>

Successor function

<b>sign</b> ° <i>int</i>	--	<i>int</i>
<b>sign</b> ° <i>real</i>	--	<i>real</i>

Sign function

<b>abs</b> ° <i>int</i>	--	<i>int</i>
<b>abs</b> ° <i>real</i>	--	<i>real</i>

Absolute value function

<b>neg</b> ° <i>int</i>	--	<i>int</i>
<b>neg</b> ° <i>real</i>	--	<i>real</i>
<b>_</b> ° <i>int</i>	--	<i>int</i>
<b>_</b> ° <i>real</i>	--	<i>real</i>

Negation of a number.

<b>round</b> ° <i>num</i>	--	<i>int</i>
<b>round</b> ° <i>complex</i>	--	<i>int</i>

Rounding to an integer.

<b>trunc</b> ° <i>num</i>	--	<i>int</i>
---------------------------	----	------------

Truncate to an integer.

<b>int</b> ° <i>num</i>	--	<i>real</i>
-------------------------	----	-------------

Integer part of the number as a real number.

<b>frac</b> ° <i>num</i>	--	<i>real</i>
--------------------------	----	-------------

Fraction part of a real number.

<b>float</b> ° <i>num</i>	--	<i>real</i>
<b>float</b> ° <i>complex</i>	--	<i>real</i>

Conversion to the real number.

<i>num</i> <b>roundto</b> <i>num</i>	--	<i>real</i>
--------------------------------------	----	-------------

<b>exp</b> ° <i>real</i>	--	<i>real</i>
--------------------------	----	-------------

Exponential function

<b>ln</b> ° <i>real</i>	--	<i>real</i>
-------------------------	----	-------------

Natural logarithm.

<b>lg</b> ° <i>real</i>	--	<i>real</i>
-------------------------	----	-------------

Decadic logarithm.

<b>ld</b> ° <i>real</i>	--	<i>real</i>
-------------------------	----	-------------

Binary logarithm.

<b>sq</b> ° <i>int</i>	--	<i>int</i>
<b>sq</b> ° <i>real</i>	--	<i>real</i>

Square of a number.

<b>sqrt</b> ° <i>num</i>	--	<i>real</i>
--------------------------	----	-------------

Square root of a number.

<b>pi</b>	--	3.141592653589793
-----------	----	-------------------

Ludolph's number  $\pi$

<b>2pi</b>	--	6.283185307179586
Scope of the unit circle.		
<b>sin</b> ° <i>real</i>	--	<i>real</i>
Sine function		
<b>cos</b> ° <i>real</i>	--	<i>real</i>
Cosine function		
<b>tan</b> ° <i>real</i>	--	<i>real</i>
Tangent function		
<b>cot</b> ° <i>real</i>	--	<i>real</i>
Cotangent function		
<b>sec</b> ° <i>real</i>	--	<i>real</i>
<b>csc</b> ° <i>real</i>	--	<i>real</i>
<b>arcsin</b> ° <i>real</i>	--	<i>real</i>
Arcsine function		
<b>arccos</b> ° <i>real</i>	--	<i>real</i>
Arccosine function		
<b>arctan</b> ° <i>real</i>	--	<i>real</i>
Arctangent function		
<i>num</i> <b>arctan2</b> <i>num</i>	--	<i>real</i>
<b>arccot</b> ° <i>real</i>	--	<i>real</i>
<b>arcsec</b> ° <i>real</i>	--	<i>real</i>
<b>arccsc</b> ° <i>real</i>	--	<i>real</i>
<b>sinh</b> ° <i>real</i>	--	<i>real</i>
Hyperbolic sine function		
<b>cosh</b> ° <i>real</i>	--	<i>real</i>
Hyperbolic cosine function		
<b>tanh</b> ° <i>real</i>	--	<i>real</i>
Hyperbolic tangent function		
<b>coth</b> ° <i>real</i>	--	<i>real</i>
<b>sech</b> ° <i>real</i>	--	<i>real</i>
<b>csch</b> ° <i>real</i>	--	<i>real</i>

<b>arsinh</b> ° <i>real</i>	--	real	
<b>arcosh</b> ° <i>real</i>	--	real	
<b>artanh</b> ° <i>real</i>	--	real	
<b>arcoth</b> ° <i>real</i>	--	real	// Library "complex.txt"
<b>arsech</b> ° <i>real</i>	--	real	// Library "complex.txt"
<b>arcsch</b> ° <i>real</i>	--	real	// Library "complex.txt"
<i>real</i> <b>root</b> <i>real</i>	--	real	// Library "complex.txt"
<b>deg</b> ° <i>num</i>	--	<i>real</i>	
Radiant-to-Degree function			
<b>rad</b> ° <i>num</i>	--	<i>real</i>	
Degree-to-Radiant function			
<b>hex</b> ° <i>num</i>	-	<i>string</i>	
Number as hexadecimal string.			
<i>real</i> <b>mod</b> <i>real</i>	--	<i>real</i>	
Modulo of real numbers.			
<b>sum</b> ° <i>list</i>	--	<i>num</i>	
Sum of the list items.			
<b>prod</b> ° <i>list</i>	--	<i>num</i>	
Product of the list items.			
<b>avg</b> ° <i>list</i>	--	<i>real</i>	
Average value of the list items.			
integral			
dd			
<b>zero</b> ° <i>data</i>	--		
<b>one</b> ° <i>data</i>	--		
<b>half</b> ° <i>data</i>	--	...	// Library "complex.txt"



## Dictionary Functions, Operators and Combinators

*dict* is a table for pattern matching treatment

*dict* = (value0 key0 value1 key1 value2 key2 ... ...)

**\_super**

Key for the super dictionary.

*dict* **get** key                      --              value

Get the *value* for the *key* from a *dict*.

*dict* **put** key,value,              --              dict

Replaces the *value* to a *key* in the *dict*.

*dict* **iget** ident                      --              value

*dict* **iget** index                      --              value

API-Get for identical keys.

*dict* **iput** ident,value,              --              dict

*dict* **iput** index,value,              --              dict

API-Put for identical keys.

**#ident** ° *dict*                      --              value

(*ident \_v*) ° *dict*                      --              value

Instance variable value.

(*ident* := *value*) ° *dict*              --              dict

Substitution of an instance variable with a *value*.

*func* <- key1 ; key2 ; ... ;

*func* ← key1 ; key2 ; ... ;

Assign combinator, general.

*func* <- key1 ifunc1 key2 ifunc2 ... ...

*func* ← key1 ifunc1 key2 ifunc2 ... ...

Assign combinator, typed.

**keys** ° *dict*                      --              list

**values** ° *dict*                      --              list

## Boolean Functions and Operators

*bool* = **true** or **false**

**'true'**                      --        *bool*  
Value for true.

**'false'**                     --        *bool*  
Value for false.

*data* = *data*                --        *bool*  
Check for equality.

*data* <> *data*              --        *bool*  
*data* != *data*              --        *bool*  
*data* ≠ *data*                --        *bool*  
Check for inequality.

*data* < *data*                --        *bool*  
Checks whether smaller.

*data* > *data*                --        *bool*  
Checks whether larger.

*data* <= *data*              --        *bool*  
Checks whether less than or equal.

*data* >= *data*              --        *bool*  
Checks whether greater than or equal to.

¬ ° *bool*                    --        *bool*  
**not** ° *bool*                --        *bool*  
**not** ° *int*                  --        *int*  
NOT function

*bool* **and** *bool*            --        *bool*  
*int* **and** *int*              --        *int*  
AND operator

*bool* **or** *bool*             --        *bool*  
*int* **or** *int*                --        *int*  
OR operator

*bool* **xor** *bool*            --        *bool*  
*int* **xor** *int*              --        *int*  
Exclusive-OR operator

**isatom** ° *data*                --        *bool*

Checks whether the *data* is a basic data type. (?)

**isprop** ° *data*                --        *bool*

Checks whether the *data* is a triple value. (?)

**islist** ° *data*                --        *bool*

Checks whether the *data* is a list.

**isbool** ° *data*                --        *bool*

Checks whether the *data* is a Boolean identifier.

**isnum** ° *data*                --        *bool*

Checks whether the *data* is a number. Generic function.

**iszero** ° *data*                --        *bool*

Checks whether the *data* is zero. Generic function.

**ispos** ° *data*                --        *bool*

Checks whether the *data* is greater than zero. Generic function.

**isneg** ° *data*                --        *bool*

Checks whether the *data* is less than zero. Generic function.

**isnil** (?)

**ispreg** (?)

**isnull** ° *data*                --        *bool*

**isint** ° *data*                --        *bool*

**isreal** ° *data*                --        *bool*

**isstring** ° *data*                --        *bool*

**isident** ° *data*                --        *bool*

**isprefix** ° *data*                --        *bool*

**isindex** ° *data*                --        *bool*

**isarray** ° *data*                --        *bool*

**iscons** ° *data*                --        *bool*

**iscombi** ° *data*                --        *bool*

**isalt** ° *data*                --        *bool*

**isobj** ° *data*                --        *bool*

**isquote** ° *data*                --        *bool*

**isivar** ° *data*                --        *bool*

**isact** ° *data*                --        *bool*

Predicates to check the appropriate data type.

**isbound** ° *ident*                --        *bool*

**isbound** ° *prefix*                --        *bool*

Checks whether an identifier is bound.

**isundef** ° *data*            --        *bool*  
 Testing for `_undef`

**iscomplex** ° *complex*       --        *bool*  
 Checks whether it is a complex number. (?)

**ismatrix** ° *object*            --        *bool*

**isodd** ° *int*                    --        *bool*  
**isodd** ° *real*                    --        *bool*

*object is ident*                --        *bool*  
 Checks whether the *ident* is the same as the class identifier of the *object*. (?)

(*ident hastag*) ° *data*        --        *bool*     (?name)

## Combinators for Program Execution (?)

*combi* = (*term \_combine* .. *arg*)

*func \_s*  
 Single function evaluation

' *literal*  
*literal k*  
*literal \_q*  
 Constant combinator

*f* : *x*  
 Application                    // to be used for closed and lift

*func1* ° *func2*  
*func1* ° *func2*  
*func1* ° *func2*                // unicode: 0x2218  
 Composition of functions.

*functional app* *argument*  
 Apply operator

*func1* , *func2* , *func3* , ... ,  
 Construction of lists.

*test* -> *then* | *else*  
*test* → *then* | *else*  
*test* -> *then* ; *else*  
 Condition with Alternat/Cons

*test ->\* func*

*test →\* func*

while Loop

*func loopif test*

do-while Loop

*(func do)°num,num,num,*

*functional for num,num,num,*

*list map functional*

Map operator

*(func aa)° list*

*(func α)° list*

Apply-to-all combinator

*list insl functional*

Insertl operator

*list insr functional*

Insertr operator

**fold** ° 'expr,initakku,list,                    --                    akku

*(func \)° list*

Insertr combinator

*list filter functional*

Filter operator

*(list,arg1,arg2,...,) map0 functional*

*(func aa0)° list,arg1,arg2,...,*

Combination of **aa** and **distr**, extended.

*func1 ee func2*

**ee** ° data,data,

Eval-Eval combinator for infix notation.

*func1 swee func2*

**swee** ° data,data,

Swap-Eval-Eval combinator

*(func1 eea func2)° argum                    --                    (x ; y ; argum ;)*

*(func dip)° list*

*(func dip)° object*

Dip combinator (stolen from Joy)

(test **try** then | else) ° argument  
in then/else with (testresult ; argument ;)

'expr **step** list,akku,

'expr **times** rep,akku,

ifnull

ifprop

data1 ?? data2            --       data

(func Y)

Y-Combinator...

**quote** ° data            --       func

Quote functional

func1 **comp** func2       --       func

Compose functional

(func **any**) ° list        --       bool

(func **all**) ° list        --       bool

## Misc Functions and Operators

**undef**                    --       error

Function is defined as undefined.

**id** ° argument            --       argument

Identity function.

**out** ° argument            --       argument

// \*Side effect

Output for debugging.

data **min** data            --       data

**min** ° data,data,        --       data

Minimum of two values.

data **max** data            --       data

**max** ° data,data,        --       data

Maximum of two values.

**name** ° ident            --       string

Print name of an identifier.

**body** ° *ident*                    --            *value*

The assigned *value* of an identifier.

**address** ° *data*                    --            *real*

Address value of the triple cell.

**identlist**                            --            *list*

List of all used identifiers. (?)

**indexdict**                            --            *dict*

Dict of all index types with integers.

**maxcell**                              --            *int*

**pointersize**                        --            *int*

win32 = [32], win64 = [64]

**\_reserve**

Value for an unbound identifier.

**\_undef**

Value for undefined.

**gc** ° *argument*                    --            *argument*

Turns on the garbage collector.

## String Functions and Operators

**substring** ° *string,num,num,*            --            *string*

*string* **concat** *string*                    --            *string*

*string* & *string*                    --            *string*

Concatenates the strings.

*string* **indexof** *substr*            --            *real*

*list* **join** *sepstr*                    --            *string*

*string* **split** *sepstr*                    --            *list*

*string* **repeat** *num*                    --            *string*

*string* **delete** *num,num,*            --            *string*

*string* **insert** *num,string,*            --            *string*

<b>length</b> ° <i>string</i>	--	<i>real</i>	
Length of the string.			
<i>string</i> <b>mid</b> <i>num,num,</i>	--	<i>string</i>	
<i>string</i> <b>left</b> <i>num</i>	--	<i>string</i>	
<i>string</i> <b>right</b> <i>num</i>	--	<i>string</i>	
<b>char</b> ° <i>num</i>	--	<i>string</i>	
<b>unicode</b> ° <i>string</i>	--	<i>real</i>	
<b>trim</b> ° <i>string</i>	--	<i>string</i>	
Trims the <i>string</i> on the left and right side.			
<b>triml</b> ° <i>string</i>	--	<i>string</i>	
Trims the <i>string</i> on the left.			
<b>trimr</b> ° <i>string</i>	--	<i>string</i>	
Trims the <i>string</i> on the right.			
<b>upper</b> ° <i>string</i>	--	<i>string</i>	
AnsiUpperCase of the string.			
<b>lower</b> ° <i>string</i>	--	<i>string</i>	
AnsiLowerCase of the string.			
<b>capitalize</b> ° <i>string</i>	--	<i>string</i>	
<b>parse</b> ° <i>string</i>	--	<i>list</i>	
Precompiles the <i>string</i> into a <i>list</i> .			
<b>value</b> ° <i>string</i>	--	<i>data</i>	
Converts the <i>string</i> to a <i>data</i> type.			
<b>string</b> ° <i>data</i>	--	<i>string</i>	
Converts the <i>data</i> to its text representation.			
<b>unpack</b> ° <i>string</i>	--	<i>list</i>	
Splits the <i>string</i> into a list of individual string characters.			
<b>pack</b> ° <i>list</i>	--	<i>string</i>	
Concatenates the strings in the <i>list</i> .			
<b>timetostring</b> ° <i>real</i>	--	<i>string</i>	
<b>datetostring</b> ° <i>num</i>	--	<i>string</i>	
<b>weekday</b> ° <i>num</i>	--	<i>num</i>	(?)



## OOP

```
object = (cap :: inst)           // Object classes

pair = object , parameter ,

self ° pair

para ° pair

index op func

index swop func

index fn func

(object (index cb func) parameter) ° argum -- method ° [0],[1],argum,

cap ° list           --      ( )
cap ° object         --      (cap ::)

ident obj list       --      (ident :: list)
ident obj dict       --      (ident :: dict)

ident new parameter

object as ident (?)   --      object

box ° primdata       --      object

unbox ° object       --      primdata

(func objdip) ° pair --      object

object == .. { ( ) ... .. }
Object class

list == .. { object ... .. }
List class

dict == .. { object ... .. }
Dict class
```

## Monads and Effects

*monad* = (*int \_act .. dict*)                      // absolute

*monad* = (*index \_act .. dict*)                      // relative

**it** ° *dict*                                      --                      #\_it ° *dict*

Result of a monad action.                      // monad ... name (?)

**\_it**

Attribute for results of actions.

**\_self**

Attribute for the first parameter for an action.

**\_para**

Attribute for the second parameter for an action.

**\_bind**

Attribute for the continuation term of an action.

**\_eff**

Attribute for the effects of relative actions.

*monad* >> *term*                      --                      *monad*                      // \_bind := *term*

Adds the continuation *term* to the table of the monad.

*int act dict*                                      --                      *monad*

*index act dict*                                      --                      *monad*

*monad act dict*                                      --                      *monad*

Create a monad. The monad needs an absolute address for the action/or relative address for the effect method/or a parent monad. In addition, the monad needs a table with instance variables. The instance variables are passed on to the continuation after the action. (nested act possible)

*monad eff array*                                      --                      *monad*

*monad eff ident*                                      --                      *monad*

Adds the effects class to the monad's table.

*monad var data*                                      --                      *monad*

*monad var dict*                                      --                      *monad*

(*ident define data*) ° *dict*                                      --                      *monad*

//(prefix define data) ° *dict*

(*ident redefine data*) ° *dict*                                      --                      *monad*

//(prefix redefine data) ° *dict*

(*data showgraph*) ° *dict*                                      --                      *monad*                      // \*+ (x eff 'io)

(*data showinfo*) ° *dict*                                      --                      *monad*                      // \*+ (x eff 'io)

```

(data print) ° dict           --      monad           // *+ (x eff 'io)

(string input) ° dict         --      monad           // *+ (x eff 'io)
(string input string) ° dict

(fname loadtext) ° dict       --      monad           // *+ (x eff 'io)
Loads a string from a text file with path and filename of fname.
The string can be obtained from #_it .

(fname savetext string) ° dict --      monad           // *+ (x eff 'io)

(string run) ° dict           --      monad           // *+ (x eff 'io)

quit                        --      monad

time ° dict                 --      monad           // *+ (x eff 'io)

date ° dict                 --      monad           // *+ (x eff 'io)

beep ° dict                 --      monad           // *+ (x eff 'io)

io == .. { ... .. }
System effects class

(etwas zu Algebraischen Effekten schreiben, [name], io-klasse/treiber, etc)

```

## Runtime Errors(?)

```

error = (index _error string ; ... ..)

index error string,         --      error

fail ° argument             --      error
Use for selector signatures(?)

stop ° argument             --      error
Generally, e.g. Program termination, etc

raise ° string              --      exception
An exception is thrown.

_error == .. { ... .. }
Class for redirects...

// try

```

## Complex Numbers

```
complex = (complex :: real re real im)           // Library "complex.txt"

i          --      (complex :: 0 re 1 im)
Square root of _1

real j real          --      complex           // für schnelle Schreibweise

real cval real       --      complex

To form a complex number from real numbers.

re ° complex          --      real
Real part of the complex number.

im ° complex          --      real
Imaginary part of the complex number.

complex + complex     --      complex
Addition of complex numbers.

complex - complex     --      complex
Subtraction of complex numbers.

complex * complex     --      complex
complex × complex     --      complex
Multiplication of complex numbers.

complex / complex     --      complex
complex ÷ complex     --      complex
Division of complex numbers.

zero ° complex        --      (complex :: 0 re 0 im)

one ° complex         --      (complex :: 1 re 0 im)

half ° complex        --      (complex :: 0.5 re 0 im)

iszero ° complex      --      bool

isnum ° complex       --      true

complex = complex     --      bool

conj ° complex        --      complex

neg ° complex         --      complex

abs ° complex         --      real

phase ° complex       --      real           // wie Arg(z)
```

<b>sq</b> ° <i>complex</i>	--	complex	
<b>exp</b> ° <i>complex</i>	--	complex	
<b>ln</b> ° <i>complex</i>	--	complex	// Hauptzweig
<b>lg</b> ° <i>complex</i>	--	complex	// Log10(z)
<b>ld</b> ° <i>complex</i>	--	complex	// Log2(z)
<i>complex</i> ^ <i>complex</i>	--	complex	
<i>complex</i> <b>root</b> <i>complex</i>	--	complex	
<b>sqrt</b> ° <i>complex</i>	--	complex	
<b>sin</b> ° <i>complex</i>	--	complex	
<b>cos</b> ° <i>complex</i>	--	complex	
<b>tan</b> ° <i>complex</i>	--	complex	
<b>cot</b> ° <i>complex</i>	--	complex	
<b>sec</b> ° <i>complex</i>	--	complex	
<b>csc</b> ° <i>complex</i>	--	complex	
<b>arcsin</b> ° <i>complex</i>	--	complex	
<b>arccos</b> ° <i>complex</i>	--	complex	
<b>arctan</b> ° <i>complex</i>	--	complex	
<b>arccot</b> ° <i>complex</i>	--	complex	
<b>arcsec</b> ° <i>complex</i>	--	complex	
<b>arccsc</b> ° <i>complex</i>	--	complex	
<b>sinh</b> ° <i>complex</i>	--	complex	
<b>cosh</b> ° <i>complex</i>	--	complex	
<b>tanh</b> ° <i>complex</i>	--	complex	
<b>coth</b> ° <i>complex</i>	--	complex	
<b>sech</b> ° <i>complex</i>	--	complex	
<b>csch</b> ° <i>complex</i>	--	complex	

```

arsinh ° complex      --      complex
arcosh ° complex      --      complex
artanh ° complex      --      complex
arcoth ° complex      --      complex
arsech ° complex      --      complex
arcsch ° complex      --      complex
iscomplex ° object    --      bool

complex == .. { dict ... .. }
Complex-class with the complex methods.

```

## Matrix Functions and Operators

```

matrix = (list ; list ; ... ;) oder
matrix = (matrix :: list ; list ; ... ;)           // Library "matrix.txt"

IP ° list,list,                                     // Backus Turing Lecture
list IP list

MM ° matrix,matrix,                                  // Backus Turing Lecture
matrix MM matrix

det ° matrix      --      real

inv ° matrix      --      matrix

matrix + matrix      --      matrix

matrix - matrix      --      matrix

matrix * matrix      --      matrix
matrix × matrix      --      matrix

trans ° matrix      --      matrix           // transpose

num scale matrix      --      matrix           // (?) scalar

matrix each 'func      --      matrix           // (?)

sq ° matrix      --      matrix

```

```

zero ° matrix          --      matrix          // zeromatrix

one ° matrix           --      matrix          // idmatrix

num0 zeromatrix num1  --      matrix
Creates a num0×num1 matrix of zero values with the type of num0.

num0 onematrix num1  --      matrix
Creates a num0×num1 matrix of one values with the type of num0.

idmatrix ° num         --      matrix
Creates a num×num identity matrix with the type of num.

ismatrix ° data        --      bool

tomatrix ° list         --      matrix

num like data          --      num            // with type of data

matrix == .. { list ... .. }
Matrix-class for MM (*), det, inv, trans, add, sub, Aij, negifodd

```

## Turtle Graphics

```

turtle = ( turtle :: list stack real x real y real angle
            bool pen num color num size num brush ) // Library "turtlegraphics.txt"

pair = (x , y ,)

// 2pi

initturtle
'turtle new           // recommended

pair moveto turtle

pair moverel turtle

real move turtle

real turnto turtle

real turn turtle

penup ° turtle

pendown ° turtle

```

*num* **pencolor** *turtle*

*num* **pensize** *turtle*

*num* **brushcolor** *turtle*

*real* **circle** *turtle*

**rectangle** ° *turtle*        // rect

(*turtle* (**draw** eff 'io)) ° *dict*        --        monad  
For drawing the turtle trail.

**#x** ° *turtle*        --        real

**#y** ° *turtle*        --        real

**#angle** ° *turtle*        --        real

etc

Attributes of the turtle object.

**colors** == '(... ...)

**#red** ° *colors*        for the color value red.

**turtle** == .. { *dict* ... ... }

Turtle class,

own turtle classes can also be created through inheritance.

(*xlist* (**plot0** eff 'io) 0-y) ° *dict*        --        monad