

## FP trivia Language Reference

2021-02-15

### 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* ≡

## 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*<sub>1</sub> *infix*<sub>1</sub> *first*<sub>2</sub> *infix*<sub>2</sub> ... ... *first*<sub>*m*</sub> *infix*<sub>*m*</sub>)

*list* = (*element*<sub>0</sub> ; *element*<sub>1</sub> ; *element*<sub>2</sub> ; ... ; )

[ *i* ] ° *list*      --      *element*<sub>*i*</sub>

**head** ° *dict*      --      *first*

**head** ° *list*      --      *first*

First element of the list.

**head** ° *object* --  
**tail** ° *dict* -- *rest*  
**tail** ° *list* -- *rest*  
 List without the first element and first infix.  
**tail** ° *object* --  
**infix** ° *dict* -- infix value  
**infix** ° *object* --  
**prop** ° *first, infix, rest,* -- *dict*  
**top** ° *dict* -- *first*  
**top** ° *list* -- *first*  
**pop** ° *dict* -- *rest*  
**pop** ° *list* -- *rest*  
**tag** ° *data* -- *typus* // als *typeof*  
**tag** ° *dict* -- infix value  
**term** ° *combi* -- term value  
**arg** ° *combi* -- arg value  
 termoarg -- term o arg  
*first , rest* -- list  
 Appendleft  
**length** ° *dict* -- *real*  
**length** ° *list* -- *real*  
 Number of list items.  
 length ° *object*  
**reverse** ° *dict* -- *dict*  
**reverse** ° *list* -- *list*  
**reverse** ° *object* --  
 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*

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

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

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

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

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

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

**1** ° *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*

*int* **downto** *int*        --        *list*

*real* **downto** *real*      --      list  
**swap** ° *x,y,list*      --      *y,x,list*

## Math Functions and Operators

*int* + *int*      --      *int*  
*real* + *real*      --      *real*  
 Addition of numbers.

*int* - *int*      --      *int*  
*real* - *real*      --      *real*  
 Subtraction of numbers.

*int* \* *int*      --      *int*  
*real* \* *real*      --      *real*  
*int* × *int*      --      *int*  
*real* × *real*      --      *real*  
 Multiplication of numbers.

*num* / *num*      --      *real*  
*num* ÷ *num*      --      *real*  
 Division of numbers.

*int* ^ *int*      --      *int*  
*real* ^ *real*      --      *real*  
 Power of numbers.

*int* **idiv** *int*      --      *int*  
 Integer division

*int* **imod** *int*      --      *int*  
 Integer modulo

**pred** ° *int*      --      *int*  
**pred** ° *real*      --      *real*  
 Predecessor function

**succ** ° *int*      --      *int*  
**succ** ° *real*      --      *real*  
 Successor function

**sign** ° *int*      --      *int*

**sign** ° *real*    --    *real*

Sign function

**abs** ° *int*        --    *int*

**abs** ° *real*       --    *real*

Absolute value function

**neg** ° *int*        --    *int*

**neg** ° *real*       --    *real*

**\_** ° *int*            --    *int*

**\_** ° *real*           --    *real*

Negation of a number.

**round** ° *num*    --    *int*

Rounding to an integer.

**trunc** ° *num*    --    *int*

Truncate to an integer.

**int** ° *num*        --    *real*

Integer part of the number as a real number.

**frac** ° *num*       --    *real*

Fraction part of a real number.

**float** ° *num*    --    *real*

Conversion to the real number.

*num* **roundto** *num*    --    *real*

**exp** ° *real*       --    *real*

Exponential function

**ln** ° *real*        --    *real*

Natural logarithm.

**lg** ° *real*        --    *real*

Decadic logarithm.

**ld** ° *real*        --    *real*

Binary logarithm.

**sq** ° *int*        --    *int*

**sq** ° *real*        --    *real*

Square of a number.

**sqrt** ° *num*    --    *real*

Square root of a number.

**pi**                --    3.141592653589793

Ludolph's number:  $\pi = 3.14159265358979323846264338327950288...$

**2pi**              --    6.283185307179586

Scope of the unit circle.

**sin** ° *real*        --    *real*

Sine function

**cos** ° *real*        --    *real*

Cosine function

**tan** ° *real*        --    *real*

Tangent function

**arcsin** ° *real*    --    *real*

Arcsine function

**arccos** ° *real*    --    *real*

Arccosine function

**arctan** ° *real*    --    *real*

Arctangent function

*num* **arctan2** *num*    --    *real*

**sinh** ° *real*        --    *real*

Hyperbolic sine function

**cosh** ° *real*        --    *real*

Hyperbolic cosine function

**tanh** ° *real*        --    *real*

Hyperbolic tangent function

**arsinh** ° *real*    --    *real*

**arcosh** ° *real*    --    *real*

**artanh** ° *real*    --    *real*

**deg** ° *num*        --    *real*

Radiant-to-Degree function

**rad** ° *num*      --      *real*

Degree-to-Radiant function

*real* **mod** *real* --      *real*

Modulo of real numbers.

**sum** ° *list*      --      *num*

Sum of the list items.

**prod** ° *list*      --      *num*

Product of the list items.

**avg** ° *list*      --      *real*

Average value of the list items.

integral

dd

## Dictionary 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 isfunc1 key2 isfunc2 ... ..*

*func ← key1 isfunc1 key2 isfunc2 ... ..*

Assign combinator, typed.

## 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*

<b>isreal</b> ° <i>data</i>	--	<i>bool</i>
<b>isstring</b> ° <i>data</i>	--	<i>bool</i>
<b>isident</b> ° <i>data</i>	--	<i>bool</i>
<b>isprefix</b> ° <i>data</i>	--	<i>bool</i>
<b>isindex</b> ° <i>data</i>	--	<i>bool</i>
<b>isarray</b> ° <i>data</i>	--	<i>bool</i>
<b>iscons</b> ° <i>data</i>	--	<i>bool</i>
<b>iscombi</b> ° <i>data</i>	--	<i>bool</i>
<b>isalt</b> ° <i>data</i>	--	<i>bool</i>
<b>isobj</b> ° <i>data</i>	--	<i>bool</i>
<b>isquote</b> ° <i>data</i>	--	<i>bool</i>
<b>isivar</b> ° <i>data</i>	--	<i>bool</i>
<b>isact</b> ° <i>data</i>	--	<i>bool</i>

Predicates to check the appropriate data type.

<b>isbound</b> ° <i>ident</i>	--	<i>bool</i>
<b>isbound</b> ° <i>prefix</i>	--	<i>bool</i>

Checks whether an identifier is bound.

<b>isundef</b> ° <i>data</i>	--	<i>bool</i>
------------------------------	----	-------------

Testing for `_undef`

<b>iscomplex</b> ° <i>complex</i>	--	<i>bool</i>
-----------------------------------	----	-------------

Checks whether it is a complex number. (?)

<b>isvector</b> ° <i>data</i>	--	<i>bool</i>
-------------------------------	----	-------------

<i>object</i> <b>is</b> <i>ident</i>	--	<i>bool</i>
--------------------------------------	----	-------------

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

<i>(ident</i> <b>hastag</b> ) ° <i>data</i>	--	<i>bool</i>	(?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

$func1 \circ func2$

$func1 \bullet func2$

$func1 \circ func2$

Composition of functions.

*functional* **app** *argument*

Apply operator

$func1, func2, func3, \dots,$

Construction of lists.

$test \rightarrow then \mid else$

$test \rightarrow then \mid else$

$test \rightarrow then ; else$

Condition with Alternat/Cons

$test \rightarrow * func$

$test \rightarrow * func$

while Loop

$func$  **loopif**  $test$

do-while Loop

$(func \text{ do})^{\circ} num, num, num,$

*functional* **for**  $num, num, num,$

$list$  **map** *functional*

Map operator

$(func \text{ aa})^{\circ} list$

$(func \alpha)^{\circ} list$

Apply-to-all combinator

$list$  **insl** *functional*

Insertl operator

$list$  **insr** *functional*

Insertr operator

*(func \) ° list*

Inserttr 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)

ifnull

ifprop

*data1 ?? data2*    --    *data*

*(func Y)*

Y-Combinator...

**quote** ° *data*    --    *func*

Quote functional

*func1 comp func2*    --    *func*

Compose functional

## 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*

**\_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*

<i>string</i> <b>concat</b> <i>string</i>	--	<i>string</i>
<i>string</i> <b>&amp;</b> <i>string</i>	--	<i>string</i>

Concatenates the strings.

<i>string</i> <b>indexof</b> <i>substr</i>	--	real
<i>list</i> <b>join</b> <i>sepstr</i>	--	string
<i>string</i> <b>split</b> <i>sepstr</i>	--	list
<i>string</i> <b>repeat</b> <i>num</i>	--	string
<i>string</i> <b>delete</b> <i>num,num,</i>	--	string
<i>string</i> <b>insert</b> <i>num,string,</i>	--	string
<b>length</b> ° <i>string</i>	--	real

Length of the string.

<i>string</i> <b>mid</b> <i>num,num,</i>	--	string
<i>string</i> <b>left</b> <i>num</i>	--	string
<i>string</i> <b>right</b> <i>num</i>	--	string
<b>char</b> ° <i>num</i>	--	string
<b>unicode</b> ° <i>string</i>	--	real
<b>trim</b> ° <i>string</i>	--	<i>string</i>

Trims the *string* on the left and right side.

<b>triml</b> ° <i>string</i>	--	<i>string</i>
------------------------------	----	---------------

Trims the *string* on the left.

<b>trimr</b> ° <i>string</i>	--	<i>string</i>
------------------------------	----	---------------

Trims the *string* 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>	--	string
<b>parse</b> ° <i>string</i>	--	list

Precompiles the *string* into a *list*.

**value** ° *string*            --        *data*

Converts the *string* to a *data* type.

**string** ° *data*            --        *string*

Converts the *data* to its text representation.

**unpack** ° *string*        --        *list*

Splits the *string* into a list of individual string characters.

**pack** ° *list*            --        *string*

Concatenates the strings in the *list*.

## 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*



**object** == .. { ( ) ... .. }

Object class

**list** == .. { object ... .. }

List class

**dict** == .. { object ... .. }

Dict class

**vector** == .. { ... .. }

Vector class

## Monads and Effects

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

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

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

Result of a monad action.

#\_it

#\_self

#\_para

\_bind

Continuation

\_eff

Effects

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

*int* **act** *dict*                    --            *monad*

*index* **act** *dict*                    --            *monad*

*monad*(?) **act** *dict*                    --            *monad*

*monad* **eff** *array*                    --            *monad*

*monad* **eff** *ident*                    --            *monad*

*monad* **var** *data*                    --            *monad*

*monad* **var** *dict*                    --            *monad*

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

```

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

quit                          --      monad

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

```

## 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...

```

## Complex Numbers

```

complex = (complex :: real re real im)

i      --      (complex :: 0 re 1 im)

```

Square root of  $-1$

*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.

etc

**complex** == .. { dict ... .. }

Complex-class with the complex methods.

## Matrices Functions and Operators

*matrix* = (*list* ; *list* ; ... ;)

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

*list* **IP** *list*

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

*matrix* **MM** *matrix*

**det** ° *matrix* -- real

**inv** ° *matrix* -- matrix

<b>transpose</b> ° <i>matrix</i>	--	matrix
<b>tovector</b> ° <i>list</i>	--	vector (?nötig)
<i>vector</i> + <i>vector</i>	--	vector
<i>vector</i> - <i>vector</i>	--	vector

## Turtle Graphics

*turtle* = ( **turtle** :: *list stack real x real y real angle*  
*bool pen num color num size num brush* )

*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)    --        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      --      monad

```

(CC-BY-3.0-DE) Fpstefan