

## Pointfrip Quickinfo

2022-10-07

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

### 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, dict, or array; or are integers*
name	is an identifier for a function associated with it
$_{123.5678e\_30}$	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

\*note that the constant combinator should be used.

### Definition of Functions/Constants/Operators

$identifier == term$	assigns a term to the identifier
$constname == 'literal$	Constants use the constant combinator
$operatorname == ( \dots ) \circ ee$	Operators often use an ee and [0] and [1]

## Combinators

<code>'name</code>	is the constant combinator
<code>function1 ° function2</code>	is the composition
<code>fun1 , fun2 , ... , funm ,</code>	is the construction of a list
<code>(test -&gt; then ; else)</code>	is the condition combinator with an alternative
<code>(test -&gt; * term)</code>	is a while loop
<code>(function aa)</code>	is the apply-to-all combinator (map)
<code>(function \ )</code>	is the insert combinator (reduce)
<code>function1 ee function2</code>	evaluates the functions and creates a pair from them
<code>#name</code>	picks the value for the name from a dict

## Functions

<code>id</code>	identity function
<code>iota</code>	produces a list of numbers from 1 up to the number
<code>head</code>	extracts the first value of a list
<code>tail</code>	extracts the rest of a list
<code>infix</code>	extracts the infix value of a list/dict
<code>prop</code>	creates a cell of First,Infix,Rest,
<code>top</code>	like head, but not for objects
<code>pop</code>	like tail, but not for objects
<code>tag</code>	extracts the type or infix value
<code>reverse</code>	reverses a list

<b>length</b>	returns the length of a list
<b>sin</b>	calculates the sine of a number
<b>ln</b>	calculates the natural logarithm of a number
<b>islist</b>	checks whether it is a list

## Operators

<i>first , rest</i>	Comma creates a list
<i>num1 + num2</i>	Arithmetic operators for addition,
<i>num1 - num2</i>	for subtraction
<i>num1 * num2</i>	for multiplication
<i>num1 / num2</i>	for division
<i>dict</i> <b>get</b> <i>key</i>	determines the value for the key*
<i>dict</i> <b>put</b> <i>key,value,</i>	replaces/creates a key* with value in the dict
<i>(key := value) ° dict</i>	like put, but as a value assignment in the dict
<i>num1 = num2</i>	checks for equality and then returns <b>true</b> or <b>false</b>
etc	

## Objects and Classes

<i>(turtle :: ( ) stack 0 x 0 y ...)</i>	is the turtle object with the attributes stack, x, y, ...
<i>turtle == .. { dict ... .. }</i>	is the turtle class with the ... methods

## Monads and Effects

`('turtle new) (draw eff 'io)` creates a monad for drawing the turtle track  
`io == .. { ... .. }` are the system effects, so to speak the driver (?)

et cetera can be found in the [reference/blue question mark](#)

(CC0)

\*note that the constant combinator should be used.