

# Languages-beta: SL-Funcons \*

The PPlanCompS Project

SL-Funcons.cbs | PLAIN | PRETTY

## OUTLINE

### SL-specific funcons

### Abbreviations

### Further funcons

- Binding
- Local variables
- Global bindings
- Composite input and output

---

*Language* "SL"

[ *Funcon* **sl-to-string**  
*Funcon* **integer-add-else-string-append**  
*Funcon* **int**  
*Funcon* **bool**  
*Funcon* **str**  
*Funcon* **obj**  
*Funcon* **fun**  
*Funcon* **scope-closed**  
*Funcon* **initialise-local-variables**  
*Funcon* **local-variable**  
*Funcon* **local-variable-initialise**  
*Funcon* **local-variable-assign**  
*Funcon* **initialise-global-bindings**  
*Funcon* **override-global-bindings**  
*Funcon* **global-bound**  
*Funcon* **read-line**  
*Funcon* **print-line** ]

## SL-specific funcons

*Funcon* **sl-to-string**( $V : \text{sl-values}$ ) :  $\Rightarrow \text{strings}$   
*Rule* **sl-to-string**(**null-value**)  $\rightsquigarrow$  "null"  
*Rule* **sl-to-string**( $V : \sim \text{null-type}$ )  $\rightsquigarrow$  **to-string**( $V$ )

---

\*Suggestions for improvement: [plancomps@gmail.com](mailto:plancomps@gmail.com).  
Reports of issues: <https://github.com/plancomps/CBS-beta/issues>.

```

Funcon  integer-add-else-string-append( $V_1 : \text{sl-values}$ ,  $V_2 : \text{sl-values}$ ) :  $\Rightarrow \text{sl-values}$ 
       $\rightsquigarrow$  else(
        integer-add(int  $V_1$ , int  $V_2$ ),
        string-append(sl-to-string  $V_1$ , sl-to-string  $V_2$ ))

```

## Abbreviations

```

Funcon  int( $V : \text{sl-values}$ ) :  $\Rightarrow \text{integers}$ 
       $\rightsquigarrow$  checked cast-to-type( $V$ , integers)

```

```

Funcon  bool( $V : \text{sl-values}$ ) :  $\Rightarrow \text{booleans}$ 
       $\rightsquigarrow$  checked cast-to-type( $V$ , booleans)

```

```

Funcon  str( $V : \text{sl-values}$ ) :  $\Rightarrow \text{strings}$ 
       $\rightsquigarrow$  checked cast-to-type( $V$ , strings)

```

```

Funcon  obj( $V : \text{sl-values}$ ) :  $\Rightarrow \text{objects}$ 
       $\rightsquigarrow$  checked cast-to-type( $V$ , objects)

```

```

Funcon  fun( $V : \text{values}$ ) :  $\Rightarrow \text{functions}(-, -)$ 
       $\rightsquigarrow$  checked cast-to-type( $V$ , functions( $-$ ,  $-$ ))

```

## Further funcons

Some of the funcons defined below might be sufficiently reuseful for inclusion in Funcons-beta.

### Binding

```

Funcon  scope-closed( $Env : \text{envs}$ ,  $X : \Rightarrow T$ ) :  $\Rightarrow T$ 
       $\rightsquigarrow$  closed scope( $Env$ ,  $X$ )

```

`scope-closed`( $D$ ,  $X$ ) evaluates  $D$  in the current environment, then evaluates  $X$  in the resulting environment. Note the difference between `scope-closed`( $D$ ,  $X$ ) and `closed`(`scope`( $D$ ,  $X$ )): the latter is equivalent to `closed`(`scope`(`closed`  $D$ ,  $X$ )), where  $D$  cannot reference any bindings.

### Local variables

The local variable map is stored in a variable bound to "local-variables". Initialising a local variable updates the stored local variable map. Subsequent assignments to a local variable do not change the stored map.

```

Funcon  initialise-local-variables :  $\Rightarrow \text{environments}$ 
       $\rightsquigarrow$  bind(
        "local-variables",
        allocate-initialised-variable(environments, map( )))

```

```

Funcon  local-variable( $l : \text{ids}$ ) :  $\Rightarrow \text{variables}$ 
       $\rightsquigarrow$  checked lookup(assigned bound "local-variables",  $l$ )

```

```

Funcon local-variable-initialise(I : ids, V : values) :  $\Rightarrow$  null-type
   $\rightsquigarrow$  assign(
    bound "local-variables",
    map-override(
      {I  $\mapsto$  allocate-initialised-variable(values, V)},
      assigned bound "local-variables"))

```

```

Funcon local-variable-assign(I : ids, V : values) :  $\Rightarrow$  null-type
   $\rightsquigarrow$  else(
    assign(local-variable I, V),
    local-variable-initialise(I, V))

```

## Global bindings

The global bindings map is stored in a variable bound to "global-bindings". Global declaration or redeclaration of an identifier involves updating the stored global environment.

```

Funcon initialise-global-bindings :  $\Rightarrow$  environments
   $\rightsquigarrow$  bind(
    "global-bindings",
    allocate-initialised-variable(environments, map( )))

```

```

Funcon override-global-bindings(E : environments) :  $\Rightarrow$  null-type
   $\rightsquigarrow$  assign(
    bound "global-bindings",
    map-override(E, assigned bound "global-bindings"))

```

```

Funcon global-bound(I : ids) :  $\Rightarrow$  values
   $\rightsquigarrow$  checked lookup(assigned bound "global-bindings", I)

```

## Composite input and output

```

Funcon read-line :  $\Rightarrow$  strings
   $\rightsquigarrow$  give(
    read,
    if-true-else(
      is-eq(given, '\n'),
      nil,
      cons(given, read-line)))

```

`read-line` reads characters from the standard input until a linefeed character, giving the string formed from the sequence of characters excluding the newline. If the input ends before the end of the line, it fails.

```

Funcon print-line(S : strings) :  $\Rightarrow$  null-type
   $\rightsquigarrow$  print(S, "\n")

```