msc2023

Contents

nazewnictwo

Lista rozwiewająca wątpliwości co do nazewnictwa w tym dokumencie.

- RET – przycisk *”Enter”* / *”Return”* na klawiaturze
- RMB – prawy przycisk myszy
- LMB – lewy przycisk myszi
- scheme – lispo-podobny język programowania wbudowany w program.
- `bounceable_t` – typ zapisany w języku C, który opisuje dany obiekt, z którym wiązka może się spotkać.
- implementacja (dla funkcji zdefiniowanych dla scheme) (kod pod sygnaturą funkcji) – mało ważna informacja zawierająca kod implemenujący daną funkcję.
- `document-function` – sposób dokumentowania funkcji, które zdefiniowane zostały w C.

[1] https://git.krzysckh.org/kpm/science-cup-2023/src/branch/master/scm/cdocs.scm
[2] https://git.krzysckh.org/kpm/science-cup-2023/src/branch/master/scm/colors.scm
[3] https://git.krzysckh.org/kpm/science-cup-2023/src/branch/master/scm/e.scm
[4] https://git.krzysckh.org/kpm/science-cup-2023/src/branch/master/scm/gui.scm
[5] https://git.krzysckh.org/kpm/science-cup-2023/src/branch/master/scm/interop-helpers.scm
[6] https://git.krzysckh.org/kpm/science-cup-2023/src/branch/master/scm/sel-mode.scm
[7] https://git.krzysckh.org/kpm/science-cup-2023/src/branch/master/scm/serialize.scm
[8] https://git.krzysckh.org/kpm/science-cup-2023/src/branch/master/scm/system-hooks.scm
[9] https://git.krzysckh.org/kpm/science-cup-2023/src/branch/master/scm/util.scm

co, gdzie i jak

Ten plik może być przeglądany jako dokument HTML.[10]

Główna część programu napisana jest w C, jednakże by ułatwić i usprawnić proces tworzenia, wbudowałem do niej interpreter scheme[11].

Każdy nowy plik w folderze `scm/` z rozszerzeniem `.scm` będzie linkowany z programem i uruchamiany na samym początku. Te pliki służą do definiowania "systemowych" funkcji, a więc (w przyszłości) UI, wszystkiego związanego z przemieszczaniem obiektów na ekranie, etc.

Rdzeń myślący i obliczający rzeczy zostanie jednak w C.

przykłady

Aktualną "scenę" można zapisać przez menu (RMB → `zapisz scenę do pliku` → `nazwa-pliku.scm` RET), lub wykonując funkcję (`serialize:save-to file-name`).

Wbudowane jest też kilka przykładów (zapisane w `scm/e.scm`), można je przeglądać przez menu (RMB → załaduj przykład).

jak korzystać ze scheme?

Pliki `*.scm` można "ładować" po prostu przerzucając je na działający program – jest to równoważne do (`load file-path`).

Proste wyrażenia scheme można wykonywać poprzez kliknięcie klawisze e , po wpisaniu wyrażenia wystarczy kliknąć RET i zostanie ono wyewaluowane.

cały zamysł opiera się na pomyśle *hooków* (inaczej eventów), nazwę zaciągnąłem prosto z gnu emacs. o danym hooku należy myśleć jak o evencie w JS na stronie internetowej. np.:

```
const el = document.getElementById("btn");
btn.addEventListener('click', (e) => {
  ...
});
```

tutaj wygląda tak:

```
(add-hook
 'click
 (lambda (first l r)
   ...))
```

jest wiele różnych rodzajów hooków o które można się *"zaczepić"*. pełna lista jest w `src/scheme-interop.c`.

- `'keypress` – wykonywany za każdym razem gdy klawisz na klawiaturze jest wciśnięty. jako argumenty przekazuje wciśnięty znak jako `char`, oraz jego numeryczną wartość (dla nie-ascii znaków)
- `'click` – wykonywany na każdej klatce, podczas której wciśnięty jest przycisk myszy. jako argumenty przekazuje `bool czy_pierwsze_przycisniecie`, `bool czy_lewy` i `bool czy_prawy`.
- `'unclick` – wykonywany podczas klatki w której przycisk myszy przestanie być wciskany przekazuje takie same argumenty jak `'click`
- `'resize` – wykonywany za każdym razem jak rozmiar okna zostanie zmieniony. przekazuje aktualną szerokość i wysokość okna.
- `'clocke` – wykonywany co sekundę. przekazuje aktualny czas.

[10] https://pub.krzysckh.org/msc2023.html
[11] https://tinyscheme.sourceforge.net/

- `'loge` – wykonywany za każdym `TraceLog()`iem z C, bądź `(tracelog)`iem z scheme. przekazuje typ logu i string z logiem.
- `'new` – wykonywany po stworzeniu nowego `bounceable_t` – obiektu od którego światło może się odbić. przekazuje typ stworzonego `bounceable` (jako symbol).
- `'update` – wykonywany za każdym razem, gdy dane `bounceable` zostało edytowane. przekazuje typ (j.w.) i `id`.
- `'delete` – wykonywany za każdym razem, gdy dane `bounceable` zostało usunięte. przekazuje typ (j.w.) i `id`.
- `'files-dropped` – wykonywany, gdy pliki zostały ”wrzucone” do okna (przeniesione z innego programu – drag&drop)
- `'frame` – wykonywany co klatkę. ***UWAGA: spowalnia mainloop. po wykorzystaniu należy go usunąć***

hooki dodaje się (j.w.) funkcją `add-hook`. zwraca ona `id` danego hooka, po którym można go potem usunąć. np.:

```
(define id
        (add-hook
         'frame
         (lambda ()
           (draw-text "halo" '(10 . 10) 16 white))))

(wait 2 (lambda () (delete-hook 'frame id)))
```

przez dwie sekundy będzie wyświetlać `halo` w {x: 10, y: 10}.

”własne obiekty o dowolnym kształcie”

tworzenie ”własnych obiektów o dowolnym kształcie” odbywa się poprzez zdefiniowanie ich w języku scheme. są uznawane za normalne `bounceable_t` typu `B_CUSTOM`.

wytłumaczę poprzez przykłady:

- chcę stworzyć **deltoid**, od którego wiązki światła odbijają się jak od normalnych zwierciadeł.

```
(define p1 '(100 . 100))
(define p2 '(150 . 150))
(define p3 '(100 . 400))
(define p4 '(50  . 150))
(define punkty-deltoidu (list p1 p2 p3 p4))

;; L[1-4] to linie w deltoidzie

(define L1 (list p1 p2))
(define L2 (list p2 p3))
(define L3 (list p3 p4))
(define L4 (list p4 p1))

;;       p1
;;        .
;; L4   / \   L1
;; p4 .     .  p2
;;     \   /
;; L3   \ /   L2
;;       v
;;      p3

;; punkty-deltoidu to wielokąt (poly w custom_data_t), czyli lista punktów.
```

```
;; jeśli znajdzie się w niej wiązka światła, program wykona funkcję, która
 ↪  obliczyć ma jak światło powinno się odbić.
;; w tym przypadku będzie to funkcja bounce-fn. dostaje ona jako dane punkt w
 ↪  którym wiązka światła dotknęła wielokątu,
;; oraz kąt względem osi OX
(define (bounce-fn hit-point angle)
  (let* ((hit-line
          (cond
           ((point-in-line? hit-point (car L1) (cadr L1) 2) L1) ;; na początek
 ↪  sprawdzamy o jaką linię deltoidu
           ((point-in-line? hit-point (car L2) (cadr L2) 2) L2) ;; wiązka
 ↪  światła faktycznie się odbiła i zapisujemy ją
           ((point-in-line? hit-point (car L3) (cadr L3) 2) L3) ;; jako
 ↪  hit-line
           ((point-in-line? hit-point (car L4) (cadr L4) 2) L4)
           (else
            (error "not in-line"))))
         (hit-angle (normalize-angle (angle-between (car hit-line) (cadr
 ↪  hit-line)))) ;; kąt pod jakim jest linia deltoidu
         (rel-angle (- hit-angle angle))
 ↪  ;; kąt pod jakim światło padło na deltoid
         (next-angle (normalize-angle (+ hit-angle rel-angle))))
 ↪  ;; kąt jaki teraz ma obrać światło
    (list hit-point next-angle)))
;;         ^^^^^^^^^ ^^^^^^^^^^
;;         \____                \_____
;;              \                     \
;; zwrócić mamy to samo co dostaliśmy, t.j. punkt od którego ma wiązka
 ↪  kontynuować, oraz kąt (względem osi OX)

;; musimy oczywiście też zdefiniować funkcję, która pokaże gdzie ten deltoid
 ↪  jest (t.j. narysuje go)
;; funkcja ta będzie wywoływana za każdym razem gdy klatka ma być narysowana
 ↪  (często) więc powinna być jak najkrótsza.
(define kolor-deltoidu lime-green) ; via scm/colors.scm
(define (draw-fn)
  (draw-line p1 p2 1 kolor-deltoidu)
  (draw-line p2 p3 1 kolor-deltoidu)
  (draw-line p3 p4 1 kolor-deltoidu)
  (draw-line p4 p1 1 kolor-deltoidu))

;; na koniec musimy "zarejestrować" deltoid, t.j. przekazać wszystkie
 ↪  informacje o nim programowi
(register-custom
 punkty-deltoidu
 draw-fn
 bounce-fn)
```

- chcę stworzyć portal, który przeniesie wiązkę do innego portalu o takich samych wymiarach

```
;; początek i koniec portalu mają takie same wymiary i są równoległe do osi
 ↪  OY,
;; żeby można było łatwo policzyć gdzie światło ma sie znaleźć
(define portal-start '((500 . 500) (500 . 700)))
(define portal-end   '((400 . 100) (400 . 200)))

;; portal początkowy rysowany jest kolorem zielonym, a końcowy czerwonym
```

```
(define (draw-fn)
  (draw-line (car portal-start) (cadr portal-start) 1 green)
  (draw-line (car portal-end) (cadr portal-end) 1 red))

(define (light-remap-fn hit-point angle)
  (let* ((hit-y (cdr hit-point))
         (diff-y (- hit-y (cdr (car portal-start)))) ;; różnica między
 ↪  początkiem (górą) portalu, a miejscem, gdzie wiązka go dotknęła
         (end-y (+ (cdr (car portal-end)) diff-y)))  ;; finalne y, w którym
 ↪  pojawić ma się wiązka
    (list (cons (caar portal-end) end-y) angle)))
;;               ^^^^^^^^^^^^^^^^  ^^^^^
;;        x portalu końcowego i (y portalu końcowego + diff-y)

(register-custom
 portal-start
 draw-fn
 light-remap-fn)
```

dokumentacja pre-definiowanych funkcji dla scheme:

scm/cdocs.scm[12]

(time-since-init)

zwraca ile czasu minęło od początku działania programu (wg. raylib - od `InitWindow()`)

(time)

zwraca aktualny unix timestamp

(system s)

wykonuje `sh -c $s` i zwraca stdout

argumenty

- s

(exit . status)

kończy program. zwraca `status` jeśli podany, inaczej 0

argumenty

- status

(loads s)

wykonuje `s` (to samo co eval, tylko że nie zwraca wartości i akceptuje string, nie sexp)

argumenty

- s

[12] https://git.krzyskh.org/kpm/science-cup-2023/src/branch/master/scm/cdocs.scm

(delete-hook sym n)

usuwa hook dla `sym` o id `n`

argumenty

- sym - `hookable_event_t` via src/scheme-interop.c
- n - id zwrócone przez `add-hook`

(get-source n)

zwraca informacje o źródle n

argumenty

- n

(get-all-sources)

zwraca listę wszystkich źródeł

(create-mirror x1 y1 x2 y2)

tworzy zwierciadło

argumenty

- x1
- y1
- x2
- y2

(get-mouse-position)

zwraca pozycje myszki na oknie w postaci `(x . y)`

(get-screen-size)

zwraca wielkość okna `(w . h)`

(get-winconf)

zwraca obecny winconf w postaci jak argumenty do `set-winconf`

(set-winconf bgcolor mirror-color)

ustawia winconf

argumenty

- bgcolor - kolor tła w formacie `(r g b a)` *(można pominąć* `a`*)*
- mirror-color - kolor zwierciadła w formacie j.w.

(real-tracelog t s)

wykonuje TraceLog(T, s)

argumenty

- t
- s

(real-fill-rect x y w h color)

lepiej uzywać `fill-rect`

argumenty

- x
- y
- w
- h
- color

(set-window-flag flag v)

ustawia flagę raylib

argumenty

- flag - flaga zdefiniowana w `interop-helpers.scm` jako `FLAG-*`
- v - `#t | #f`

(get-window-flag flag)

getter dla flagi raylib

argumenty

- flag - flaga zdefiniowana w `interop-helpers.scm` jako `FLAG-*`

(rect-collision r1 r2)

zwraca wspólny prostokąt dla r1 i r2. w razie braku, zwraca `(0 0 0 0)`

argumenty

- r1
- r2

(get-bounceable id)

zwraca dane dla `bounceable_t` od id `id`

argumenty

- id

(get-all-bounceables)

(set-mirror! id pt1 pt2)

zmienia dane zwierciadła o id `id`

argumenty

- id
- pt1
- pt2

(real-register-custom pts f1 f2)

patrz: register-custom

argumenty

- pts
- f1
- f2

(create-prism pt vert-len n)

tworzy pryzmat ze środkiem `pt`, długością boku `vert-len` i magicznym numerkiem `n` (wyleciało mi teraz z głowy jak to się nazywa lol)

argumenty

- pt
- vert-len
- n

(normalize-rectangle rect)

zwraca *unormalniony* prostokąt

argumenty

- rect

przykłady

`(normalize-rectangle '(10 10 -10 -10))` → *'(0 0 10 10)*

(point-in-line? pt lp1 lp2 thr)

robi raylibowe CheckCollisionPointLine(pt, lp1, lp2, thr)

argumenty

- pt
- lp1
- lp2
- thr

(angle-between p1 p2)

zwraca kąt pomiędzy `p1` a `p2` (w stopniach)

argumenty

- p1
- p2

(normalize-angle ang)

zwraca *unormalniony* kąt

argumenty

- ang

przykłady

`(normalize-angle 380)` → *20*

(vec-move-towards vec target maxlen)

Vector2MoveTowards(vec, target, maxlen)

argumenty

- vec
- target
- maxlen

(real-delete-all-sources)

(create-lens center d r)

argumenty

- center
- d
- r

(delete-bounceable id)

argumenty

- id

(set-lens! id center d r)

argumenty

- id
- center
- d
- r

(point-in-lens? pt lens-id)

argumenty

- pt
- lens-id

(white? color)

sprawdza czy kolor jest rozumiany za biały

argumenty

- color

(getenv s)

`man 3 getenv`

argumenty

- s

scm/colors.scm[13]

scm/e.scm[14]

(e:delete-all)

(load-example n)

argumenty

- n

(define-example nam user-f)

argumenty

- nam
- user-f

(rand-float)

scm/gui.scm[15]

(gui/rect rect c)

argumenty

- rect
- c

(gui/window-box-get-empty-space rect)

argumenty

- rect

(gui/window-box rect title)

rysuje bounding-box okienka wraz z tytułem, zwraca miejsce, które pozostało na elementy

argumenty

- rect - prostokąt (`x y w h`)
- title - tytuł

(gui/window-box-retained rect title)

rysuje window-box, tylko, że dodaje hooki dla 'frame. zwraca (`destruktor to-co-gui/window-box`)

argumenty

- rect
- title

[13] https://git.krzysckh.org/kpm/science-cup-2023/src/branch/master/scm/colors.scm
[14] https://git.krzysckh.org/kpm/science-cup-2023/src/branch/master/scm/e.scm
[15] https://git.krzysckh.org/kpm/science-cup-2023/src/branch/master/scm/gui.scm

(gui/input-box rect text)

argumenty

- rect
- text

(gui/label rect text)

argumenty

- rect
- text

(gui/get-max-text-length-for-width w sz)

argumenty

- w
- sz

(gui/multiline-text rect txt cursor-at)

argumenty

- rect
- txt
- cursor-at

(gui/input-popup title callback . force)

argumenty

- title
- callback
- force

(gui/message title text timeout . rect)

wyświetla wiadomość

argumenty

- title - tytuł przekazany do gui/window-box
- text - tekst wiadomości
- timeout - czas po którym wiadomość znika
- rect - rect dla wiadomości (nieobowiązkowy)

(gui/msg text)

wyświetla gui/message

argumenty

- text

(gui/option-menu user-pos opts . exit-handler)

argumenty

- user-pos - pozycja lewego-górnego punktu opcji w formie `(x . y)`
- opts - opcje w formie ((tekst . funkcja) (tekst . funkcja) ...)
- exit-handler

(gui/button-textfn rect text-fn cb . ignore-all)

tworzy przycisk w `rect` z tekstem zwróconym przez `text-fn`. po przyciśnięciu wykonuje `cb`. zwraca **destruktor** - funkcję usuwającą go

argumenty

- rect
- text-fn
- cb
- ignore-all

(gui/button rect text cb)

argumenty

- rect
- text
- cb

(gui/btn pos text cb)

wykonuje gui/button, tylko sam liczy jak szeroki i wysoki ma być przycisk się zmieścił. zwraca (destruktor szerokosc wysokosc)

argumenty

- pos - pozycja w formacie (x . y)
- text - tekst w przycisku
- cb - callback

(gui/slider rect from to cb)

tworzy slider. wywołuje `cb` z wynikiem za każdym `'unclick` eventem. zwraca destruktor.

argumenty

- rect - w formacie `(x y w h)`
- from - minimum
- to - maksimum
- cb - callback

(gui/checkbox rect cb . state)

tworzy checkbox. zwraca destruktor.

argumenty

- rect - prostokąt na checkbox
- cb - callback wykonywany po kliknięciu, jako argument przekazuje aktualną wartość (#t | #f)
- state - (opcjonalnie) początkowa wartość (#t | #f)

(gui/draw-text-persist . args)

zostawia narysowany tekst. zwraca destruktor. ***argumenty jak do*** `(draw-text)`

argumenty

- args

(gui/new-source-form)

form pytający użytkownika o dane nowego źródła

(gui/rect-fit-into-screen rect)

zwraca `rect`, który zmieści się na ekranie

argumenty

- rect

(gui/mp-slider+ok from to cb n-after-comma)

argumenty

- from
- to
- cb
- n-after-comma

(gui/show-window-opts)

(gui/show-hook-status)

(gui/show-fps pos)

argumenty

- pos

(gui/save-current)

(gui/load-example-menu)

(gui/change-source-color-form pos cb)

argumenty

- pos
- cb

scm/interop-helpers.scm[16]

(aq e alist)

zwraca wynik assq bez car

argumenty

- e - element szukany
- alist - lista asocjasyjna

[16] https://git.krzysckh.org/kpm/science-cup-2023/src/branch/master/scm/interop-helpers.scm

(cdr* l)

zwraca cdr dla l jeśli l to lista lub para

argumenty

- l

(aq-or e alist o)

zwraca wynik (assq e alist) jeśli e istnieje w alist. w przeciwnym wypadku o

argumenty

- e
- alist
- o

(update-sources)

wewnętrzna funkcja aktualizująca *sources* za każdym razem gdy zostaną zmienione

(create-source a)

tworzy nowe source_t (źródło światła)

argumenty

- a - lista asocjacyjna z elementami 'x, 'y, 'size, 'angle, 'thickness, 'reactive, 'color, 'n-beam. wszystkie elementy mają wartości domyślne i mogą być pominięte. zamiast 'x i 'y, może zostać zdefiniowane samo 'pos

przykłady

`(create-source '((x . 500) (y . 500) (reactive . #t)))` → *tworzy reagujące na myszkę źródło na pozycji (500 500)*

(draw-line pt1 pt2 thick color)

rysuje linię od pt1 do pt2 o grubości thick i kolorze color

argumenty

- pt1 - punkt 1 (x . y)
- pt2 - punkt 2 (x . y)
- thick - grubość
- color - kolor (r g b a)

(set-source! n x y ang thickness mouse-reactive n-beams color)

aktualizuje źródło o id n ustawiając wszystkie jego wartości. lepiej używać set-source-e!

argumenty

- n
- x
- y
- ang
- thickness
- mouse-reactive
- n-beams
- color

(set-source-e! n sym v)

aktualizuje właściwość sym na v w źródle o id n

argumenty

- n - id źródła
- sym - 'pos | 'angle | 'thickness | 'color | 'mouse-reactive | 'n-beams w zależności od tego co chcemy zmienić
- v - nowa wartość dla sym

(set-prism-e! id t v)

argumenty

- id
- t - `'pt` | `'vert-len` | `'n`
- v - wartość dla `t`

(set-lens-e! id t v)

argumenty

- id
- `t - r | center | d`
- v

(measure-text text size . spacing)

zwraca (w . h) tekstu text o wielkości size i spacingu spacing (jeśli podany)

argumenty

- text
- size
- spacing

(draw-text text pos sz color . spacing)

wypisuje tekst text domyślnym fontem na pozycji pos, o wielkości sz i kolorze color. można też podać spacing.

argumenty

- text
- pos
- sz
- color - w postaci (r g b a)
- spacing

(tracelog t . vs)

robi TraceLog z typem T i tekstem vs

argumenty

- t - typ logu (moze być `'trace | 'debug | 'info | 'warning | 'error | 'fatal`)
- vs - tekst

(register-custom poly-points draw-function light-remap-function)

tworzy nowy obiekt w obrębie `poly-points` rysowany co klatkę przez `draw-function`, jeśli wiązka światła napotka obiekt, przemieniana jest wg. `light-remap-function`. więcej doc TBD

argumenty

- poly-points
- draw-function
- light-remap-function

(add-user-hook s f)

w przyszłości będzie dodawała hooki które mogą być blokowane przez systemowe

argumenty

- s
- f

(stop-simulation)

zatrzymuje wszystko (przestaje renderować)

(start-simulation)

odpala z powrotem wszystko (zaczyna renderować)

(fill-rect rect color)

argumenty

- rect
- color

(experimental/toggle-resizable)

(add-mirror p1 p2)

wywołuje `create-mirror`, tylko, że argumenty to punkty w parach

argumenty

- p1 - '(x . y)
- p2 - '(x . y)

(delete-all-sources)

(delete-source id)

argumenty

- id

(delete-sources lst)

argumenty

- lst

scm/sel-mode.scm[17]

(sel-mode:highlight-rects rects sel-map)

argumenty

- rects
- sel-map

(sel-mode:get-menu)

(start-selected-mode)

(really-end-selected-mode)

scm/serialize.scm[18]

(serialize:bounceable->sexp b)

argumenty

- b

(serialize:source->sexp s)

argumenty

- s

(serialize:print sexp)

argumenty

- sexp

(serialize:save-to filename)

argumenty

- filename

(serialize:read-sexps f acc)

argumenty

- f
- acc

scm/system-hooks.scm[19]

(reposition-source-hook first left right)

argumenty

- first
- left
- right

[17] https://git.krzysckh.org/kpm/science-cup-2023/src/branch/master/scm/sel-mode.scm
[18] https://git.krzysckh.org/kpm/science-cup-2023/src/branch/master/scm/serialize.scm
[19] https://git.krzysckh.org/kpm/science-cup-2023/src/branch/master/scm/system-hooks.scm

(reposition-source-end-hook first left right)

argumenty

- first
- left
- right

(start-drawing-mirror-hook first left right)

argumenty

- first
- left
- right

(end-drawing-mirror-hook first left right)

argumenty

- first
- left
- right

(wait-handler time)

handler dla funkcji `(wait)`

argumenty

- time

(create-source-at-mouse-position)

(keypress-default-hook c _)

argumenty

- c
- _

(src->rect pos)

argumenty

- pos

(option-menu-for-lens ids)

argumenty

- ids

(option-menu-for-prism ids)

argumenty

- ids

(option-menu-for-source ids)

argumenty

- ids

(option-menu-for t id-or-ids)

argumenty

- t
- id-or-ids

(_open-menu vs)

argumenty

- vs

(display-next-log)

ale fajna funkcja ciekawe jak dziala :333

(rect-collision? r1 r2)

sprawdza czy dwa `r1` i `r2` mają punkty wspólne. zwraca `#f | #t`

argumenty

- r1
- r2

(triangle->rect p1 p2 p3)

argumenty

- p1
- p2
- p3

(prism->ptlist p)

argumenty

- p

(reposition-source-by-delta id Δ)

argumenty

- id
- Δ

(reposition-mirror-by-delta id Δ)

argumenty

- id
- Δ

(reposition-prism-by-delta id Δ)

argumenty

- id
- Δ

(reposition-lens-by-delta id Δ)

argumenty

- id
- Δ

(reposition-bounceable-by-delta id Δ)

argumenty

- id
- Δ

(lens->rect lens)

argumenty

- lens

(thing->rect thing)

argumenty

- thing

(update-toplist l id)

argumenty

- l
- id

(add-bounceable-to-toplist l id)

argumenty

- l
- id

(delete-from-toplist l id)

argumenty

- l
- id

(toggle-mode-display)

(load-files-handler . vs)

argumenty

- vs

scm/util.scm[20]

(print s . l)

wypisuje argumenty do konsoli

argumenty

- s
- l

(pprint s . l)

wypisuje argumenty do konsoli, bez spacji poiędzy

argumenty

- s
- l

(max2 a b)

zwraca większą wartość pomiędzy a, a b

argumenty

- a
- b

(min2 a b)

zwraca mniejszą wartość pomiędzy a, a b

argumenty

- a
- b

(max . ns)

zwraca największą wartość pośród argumentów

argumenty

- ns

przykłady

`(max 1 2 3)` → *3*

(min . ns)

zwraca najmniejszą wartość pośród argumentów

argumenty

- ns

przykłady

`(min 1 2 3)` → *1*

[20] https://git.krzysckh.org/kpm/science-cup-2023/src/branch/master/scm/util.scm

(maxl lst)

zwraca największą wartość z listy

argumenty

- lst

przykłady

`(max '(1 2 3))` → *3*

(minl lst)

zwraca najmniejszą wartość z listy

argumenty

- lst

przykłady

`(min '(1 2 3))` → *1*

(bool->string v)

argumenty

- v

(->string x)

zamienia cokolwiek na string

argumenty

- x

(->char x)

zamienia cokolwiek na znak

argumenty

- x

(string-split str c)

tnie *str* na każdym napodkanym *c*

argumenty

- str
- c

przykłady

`(string-split "abc|def|ghi" "|")` → *(abc def ghi)*

(filter f lst)

wywołuje f dla każdego elementu z lst i zwraca listę elementów w których f zwraca #t (lub inną nie-#f wartość)

argumenty

- f
- lst

przykłady

`(filter (lambda (v) (eq? 1 v)) '(1 2 3 1 2 5))` → *(1 1)*

(flatten lst)

zamienia zagnieżdżone listy w lst na jedną listę

argumenty

- lst

przykłady

`(flatten '(1 (2) ((3)) ((((4))))))` → *(1 2 3 4)*

(sys . l)

uruchamia `system`, wcześniej zamieniając argumenty na jeden string

argumenty

- l

(iota s step e)

generuje ciąg liczb od *d* do *e* zwiększający się o *step*

argumenty

- s
- step
- e

(sum l)

sumuje wartości listy *l*

argumenty

- l

(point-in-rect? pt rect)

sprawdza czy punkt jest w prostokącie

argumenty

- pt - punkt w postaci (x . y)
- rect - prostokąt w postaci (x y w h)

(split-every lst n)

dzieli listę *lst* co *n* elementów

argumenty

- lst
- n

(wait secs f)

wykonuje `f` po upłynięciu `secs` sekund

argumenty

- secs
- f

(last lst)

zwraca ostatni element listy

argumenty

- lst

przykłady

`(last '(a b c d))` → *d*

(avg l)

zwraca średnią z listy

argumenty

- l

(true? v)

argumenty

- v

(pts->rect p1 p2)

argumenty

- p1
- p2

(rect->poly rect)

zamienia prostokąt na listę punktów

argumenty

- rect

(round-off-zero v)

argumenty

- v

(round-off z n)

argumenty

- z
- n

przykłady

`(round-off 10.1234123 2)` → *10.12*

(get-lens-f r)

argumenty

- r

(get-all-bounceables)

(id->btype id)

argumenty

- id

(all f lst)

argumenty

- f
- lst

(all-same? l)

argumenty

- l