DSLs in Racket: You Want It How, Now?

Yunjeong Lee, Kiran Gopinathan, Ziyi Yang, Matthew Flatt, Ilya Sergey

SLE 2024

Agenda

- Language Design Intents
- Enabling Mechanisms
- Survey of DSLs in Racket

Agenda

- Language Design Intents
- Enabling Mechanisms
- Survey of DSLs in Racket

Agenda

- Language Design Intents
- Enabling Mechanisms
- Survey of DSLs in Racket

Agenda

- Language Design Intents
- Enabling Mechanisms
- Survey of DSLs in Racket

Creating a DSL in *Racket*

Creating a DSL in *Racket*

Let's create a DSL for game development!

Creating a Clickomania DSL in Racket

Let's create a DSL for developing a Clickomania game!

Clickomania game

- A 1-player logical game
- Initially populated with blocks of different colors or images
- Upon click, all adjacent blocks of the same image are removed

Creating a Clickomania DSL in Racket

Let's create a DSL for developing a Clickomania game!

Clickomania game

- A 1-player logical game
- Initially populated with blocks of different colors or images
- Upon click, all adjacent blocks of the same image are removed

Creating a Clickomania DSL in Racket

Let's create a DSL for developing a Clickomania game!

Clickomania game

- A 1-player logical game
- Initially populated with blocks of different colors or images
- Upon click, all adjacent blocks of the same image are removed

Creating a Clickomania DSL in Racket

Let's create a DSL for developing a Clickomania game!

Clickomania game

- A 1-player logical game
- Initially populated with blocks of different colors or images
- Upon click, all adjacent blocks of the same image are removed

Creating a Clickomania DSL in Racket

Let's create a DSL for developing a Clickomania game!

Clickomania game

- A 1-player logical game
- Initially populated with blocks of different colors or images
- Upon click, all adjacent blocks of the same image are removed

Creating a Clickomania DSL in Racket

Let's create a DSL for developing a Clickomania game!

Clickomania game

- A 1-player logical game
- Initially populated with blocks of different colors or images
- Upon click, all adjacent blocks of the same image are removed

Creating a Clickomania DSL in Racket

Let's create a DSL for developing a Clickomania game!

Clickomania game

- A 1-player logical game
- Initially populated with blocks of different colors or images
- Upon click, all adjacent blocks of the same image are removed

Creating a Clickomania DSL in Racket

Let's create a DSL for developing a Clickomania game!

Clickomania game

- A 1-player logical game
- Initially populated with blocks of different colors or images
- Upon click, all adjacent blocks of the same image are removed

Creating a Clickomania DSL in Racket

Let's create a DSL for developing a Clickomania game!

Clickomania game

- A 1-player logical game
- Initially populated with blocks of different colors or images
- Upon click, all adjacent blocks of the same image are removed

Creating a Clickomania DSL in Racket

Let's create a DSL for developing a Clickomania game!

Clickomania game

- A 1-player logical game
- Initially populated with blocks of different colors or images
- Upon click, all adjacent blocks of the same image are removed

Creating a Clickomania DSL in Racket

Let's create a DSL for developing a Clickomania game!

Clickomania game

- A 1-player logical game
- Initially populated with blocks of different colors or images
- Upon click, all adjacent blocks of the same image are removed

Creating a Clickomania DSL in Racket

Let's create a DSL for developing a Clickomania game!

Clickomania game

- A 1-player logical game
- Initially populated with blocks of different colors or images
- Upon click, all adjacent blocks of the same image are removed

Creating a Clickomania DSL in Racket

Let's create a DSL for developing a Clickomania game!

Clickomania game

- A 1-player logical game
- Initially populated with blocks of different colors or images
- Upon click, all adjacent blocks of the same image are removed

Creating a Clickomania DSL in Racket

Let's create a DSL for developing a Clickomania game!

Clickomania game

- A 1-player logical game
- Initially populated with blocks of different colors or images
- Upon click, all adjacent blocks of the same image are removed

Creating a Clickomania DSL in Racket

Let's create a DSL for developing a Clickomania game!

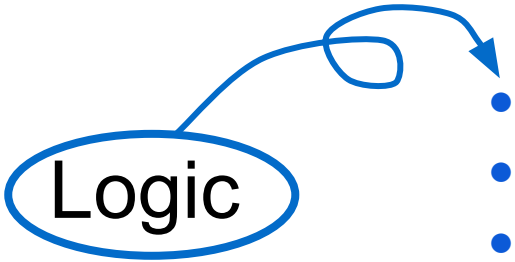
Clickomania game

- A 1-player logical game
- Initially populated with blocks of different colors or images
- Upon click, all adjacent blocks of the same image are removed

Implementing a game

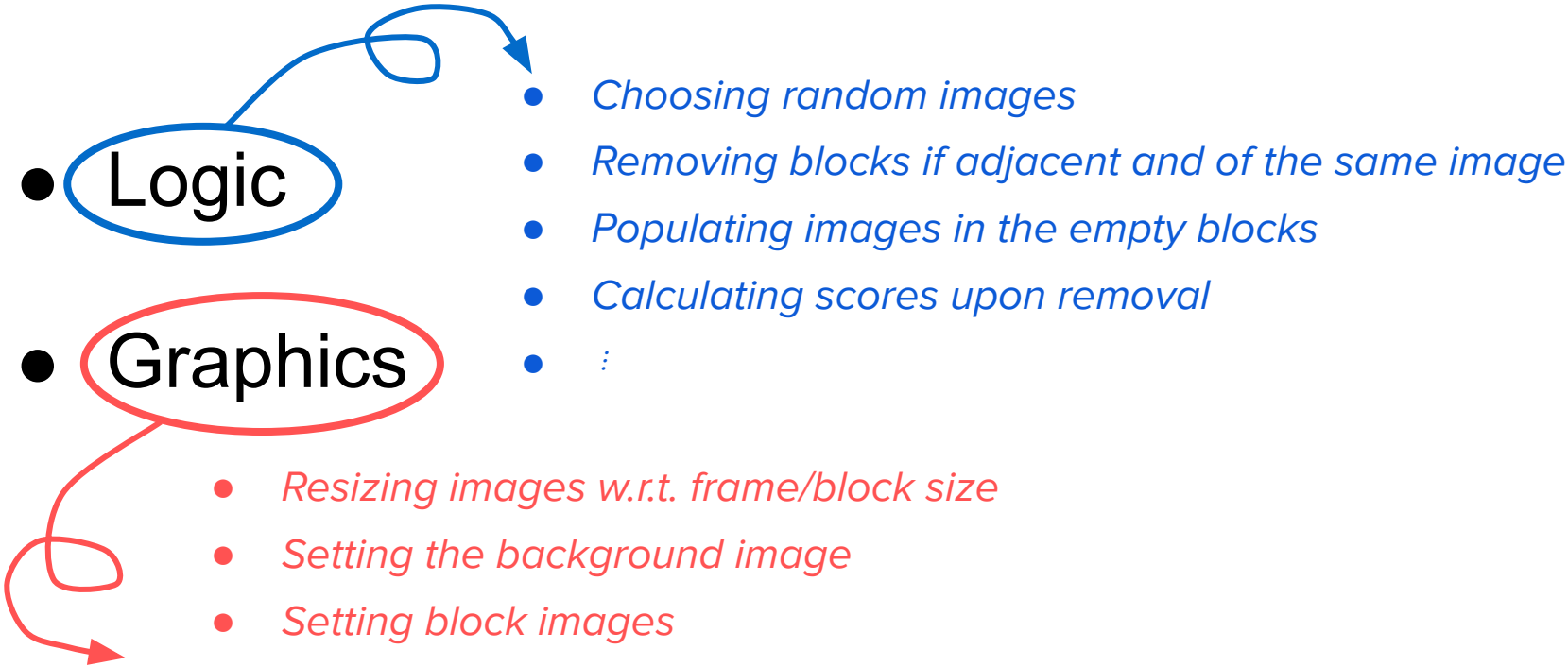
- Logic
- Graphics

Implementing a game



- Logic
  - *Choosing random images*
  - *Removing blocks if adjacent and of the same image*
  - *Populating images in the empty blocks*
  - *Calculating scores upon removal*
  - *⋮*
- Graphics

Implementing a game



```
#lang racket
(require 2htdp/image)
(define-struct wstate [position counter image remaining score])

(define curr-idxs ; (Listof (Listof Number))
  (let ([idxs (choose-rand-idxs num-colors num-blocks)])
    (lst->lstlst idxs edge-size)))

(define (negate-same-indexes-around col-row curr-idx)
  (let* ([lst (if (= -2 curr-idx)
                  (list) ; If empty (-2), return empty list
                  (list (col-row->u col-row) (col-row->d col-row)
                        (col-row->l col-row) (col-row->r col-row)))]
         ; Traverse adjacent blocks and collect col-row's to negate
         [col-rows-to-negate (collect-col-rows-to-negate
                              lst curr-idx (list col-row))]
         [num-negate-blocks (length col-rows-to-negate)])
    ; Negate indexes in curr-idxs based on col-rows-to-negate
    (for ([col-row col-rows-to-negate])
      (set-at col-row -1))
    (* num-negate-blocks block-score)))

(define (adjust-image-with-path target-size img-path)
  (let* ([img (bitmap/file img-path)]
         [img-width (image-width img)]
         [img-height (image-height img)])
    ; Return an image whose width and height are equal to targe-size
    (scale/xy (/ target-size img-width)
              (/ target-size img-height) img)))
```

```
#lang racket
(require 2htdp/image)
(define-struct wstate [position counter image remaining score])

(define curr-idxs ; (Listof (Listof Number))
  (let ([idxs (choose-rand-idxs num-colors num-blocks)])
    (lst->lstlst idxs edge-size)))

(define (negate-same-indexes-around col-row curr-idx)
  (let* ([lst (if (= -2 curr-idx)
                  (list) ; If empty (-2), return empty list
                  (list (col-row->u col-row) (col-row->d col-row)
                        (col-row->l col-row) (col-row->r col-row)))]
         ; Traverse adjacent blocks and collect col-row's to negate
         [col-rows-to-negate (collect-col-rows-to-negate
                              lst curr-idx (list col-row))]
         [num-negate-blocks (length col-rows-to-negate)])
    ; Negate indexes in curr-idxs based on col-rows-to-negate
    (for ([col-row col-rows-to-negate])
      (set-at col-row -1))
    (* num-negate-blocks block-score)))

(define (adjust-image-with-path target-size img-path)
  (let* ([img (bitmap/file img-path)]
         [img-width (image-width img)]
         [img-height (image-height img)])
    ; Return an image whose width and height are equal to targe-size
    (scale/xy (/ target-size img-width)
              (/ target-size img-height) img)))
```

```
#lang racket
(require 2htdp/image)
(define-struct wstate [position counter image remaining score])

(define curr-idxs ; (Listof (Listof Number))
  (let ([idxs (choose-rand-idxs num-colors num-blocks)])
    (lst->lstlst idxs edge-size)))

(define (negate-same-indexes-around col-row curr-idx)
  (let* ([lst (if (= -2 curr-idx)
                  (list) ; If empty (-2), return empty list
                  (list (col-row->u col-row) (col-row->d col-row)
                        (col-row->l col-row) (col-row->r col-row)))]
         ; Traverse adjacent blocks and collect col-row's to negate
         [col-rows-to-negate (collect-col-rows-to-negate
                              lst curr-idx (list col-row))]
         [num-negate-blocks (length col-rows-to-negate)])
    ; Negate indexes in curr-idxs based on col-rows-to-negate
    (for ([col-row col-rows-to-negate])
      (set-at col-row -1))
    (* num-negate-blocks block-score)))

(define (adjust-image-with-path target-size img-path)
  (let* ([img (bitmap/file img-path)]
         [img-width (image-width img)]
         [img-height (image-height img)])
    ; Return an image whose width and height are equal to targe-size
    (scale/xy (/ target-size img-width)
              (/ target-size img-height) img)))
```

```
#lang racket
(require 2htdp/image)
(define-struct wstate [position counter image remaining score])

(define curr-idxs ; (Listof (Listof Number))
  (let ([idxs (choose-rand-idxs num-colors num-blocks)])
    (lst->lstlst idxs edge-size)))

(define (negate-same-indexes-around col-row curr-idx)
  (let* ([lst (if (= -2 curr-idx)
                  (list) ; If empty (-2), return empty list
                  (list (col-row->u col-row) (col-row->d col-row)
                        (col-row->l col-row) (col-row->r col-row)))]
         ; Traverse adjacent blocks and collect col-row's to negate
         [col-rows-to-negate (collect-col-rows-to-negate
                              lst curr-idx (list col-row))]
         [num-negate-blocks (length col-rows-to-negate)])
    ; Negate indexes in curr-idxs based on col-rows-to-negate
    (for ([col-row col-rows-to-negate])
      (set-at col-row -1))
    (* num-negate-blocks block-score)))

(define (adjust-image-with-path target-size img-path)
  (let* ([img (bitmap/file img-path)]
         [img-width (image-width img)]
         [img-height (image-height img)])
    ; Return an image whose width and height are equal to targe-size
    (scale/xy (/ target-size img-width)
              (/ target-size img-height) img)))
```

```
#lang racket
(require 2htdp/image)
(define-struct wstate [position counter image remaining score])

(define curr-idxs ; (Listof (Listof Number))
  (let ([idxs (choose-rand-idxs num-colors num-blocks)])
    (lst->lstlst idxs edge-size)))

(define (negate-same-indexes-around col-row curr-idx)
  (let* ([lst (if (= -2 curr-idx)
                  (list) ; If empty (-2), return empty list
                  (list (col-row->u col-row) (col-row->d col-row)
                        (col-row->l col-row) (col-row->r col-row)))]
         ; Traverse adjacent blocks and collect col-row's to negate
         [col-rows-to-negate (collect-col-rows-to-negate
                              lst curr-idx (list col-row))]
         [num-negate-blocks (length col-rows-to-negate)])
    ; Negate indexes in curr-idxs based on col-rows-to-negate
    (for ([col-row col-rows-to-negate])
      (set-at col-row -1))
    (* num-negate-blocks block-score)))

(define (adjust-image-with-path target-size img-path)
  (let* ([img (bitmap/file img-path)]
         [img-width (image-width img)]
         [img-height (image-height img)])
    ; Return an image whose width and height are equal to targe-size
    (scale/xy (/ target-size img-width)
              (/ target-size img-height) img)))
```

```
#lang racket
(require 2htdp/image)
(define-struct wstate [position counter image remaining score])

(define curr-idxs ; (Listof (Listof Number))
  (let ([idxs (choose-rand-idxs num-colors num-blocks)])
    (lst->lstlst idxs edge-size)))

(define (negate-same-indexes-around col-row curr-idx)
  (let* ([lst (if (= -2 curr-idx)
                  (list) ; If empty (-2), return empty list
                  (list (col-row->u col-row) (col-row->d col-row)
                        (col-row->l col-row) (col-row->r col-row)))]
         ; Traverse adjacent blocks and collect col-row's to negate
         [col-rows-to-negate (collect-col-rows-to-negate
                              lst curr-idx (list col-row))]
         [num-negate-blocks (length col-rows-to-negate)])
    ; Negate indexes in curr-idxs based on col-rows-to-negate
    (for ([col-row col-rows-to-negate])
      (set-at col-row -1))
    (* num-negate-blocks block-score)))

(define (adjust-image-with-path target-size img-path)
  (let* ([img (bitmap/file img-path)]
         [img-width (image-width img)]
         [img-height (image-height img)])
    ; Return an image whose width and height are equal to targe-size
    (scale/xy (/ target-size img-width)
              (/ target-size img-height) img)))
```

They can be abstracted away … in a DSL

```
#lang racket
(require 2htdp/image)
(define-struct wstate [position counter image remaining score])

(define curr-idxs ; (Listof (Listof Number))
  (let ([idxs (choose-rand-idxs num-colors num-blocks)])
    (lst->lstlst idxs edge-size)))
```

```
#lang clickomania

(define sunset (adjust-image-with-path 500 "./sunset.png"))
(set-background-image sunset)
(run 500 5 #:theme "candy" #:num-colors 4 #:num-clicks 10)
```

```
    ; Negate indexes in curr-idxs based on col-rows-to-negate
    (for ([col-row col-rows-to-negate])
      (set-at col-row -1))
    (* num-negate-blocks block-score)))

(define (adjust-image-with-path target-size img-path)
  (let* ([img (bitmap/file img-path)]
         [img-width (image-width img)]
         [img-height (image-height img)])
    ; Return an image whose width and height are equal to targe-size
    (scale/xy (/ target-size img-width)
              (/ target-size img-height) img)))
```

```
#lang racket
(require 2htdp/image)
(define-struct wstate [position counter image remaining score])

(define curr-idxs ; (Listof (Listof Number))
  (let ([idxs (choose-rand-idxs num-colors num-blocks)])
    (lst->lstlst idxs edge-size)))
```

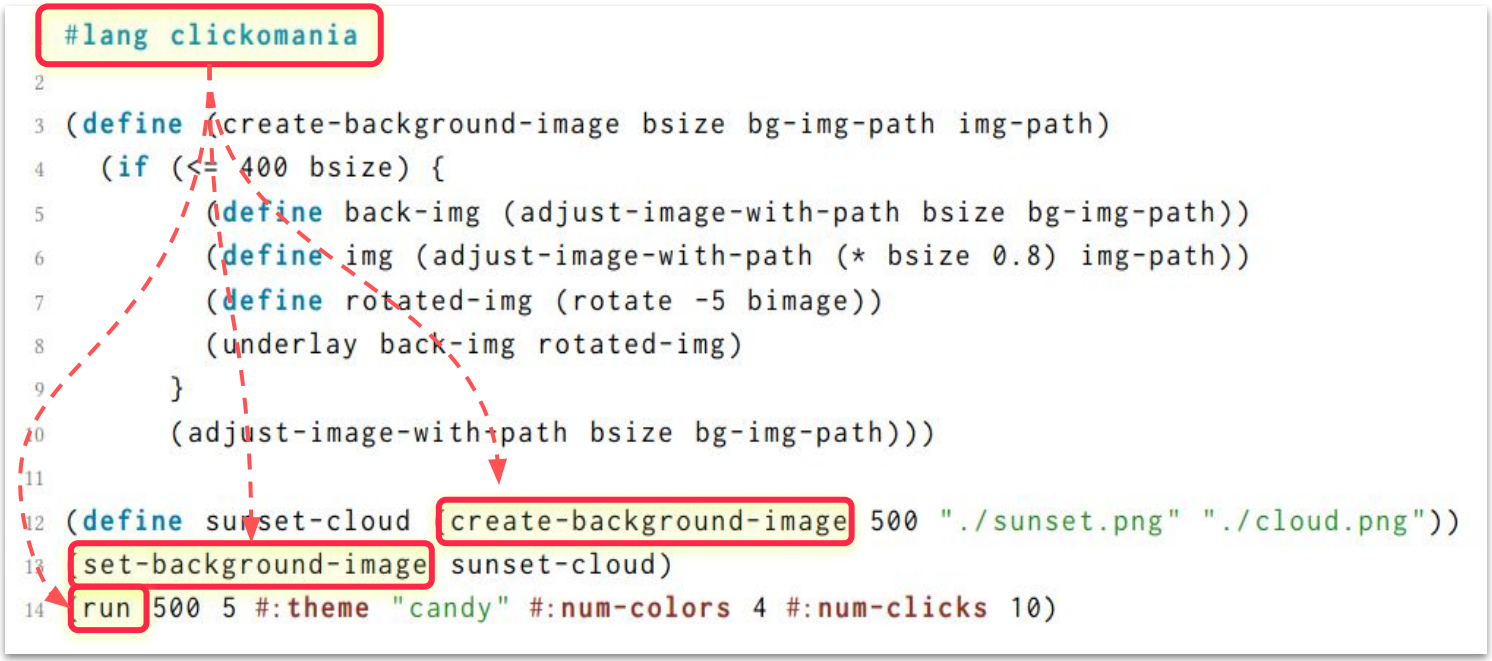
```
#lang clickomania

(define sunset (adjust-image-with-path 500 "./sunset.png"))
(set-background-image sunset)
(run 500 5 #:theme "candy" #:num-colors 4 #:num-clicks 10)
```

```
    ; Negate indexes in curr-idxs based on col-rows-to-negate
    (for ([col-row col-rows-to-negate])
      (set-at col-row -1))
    (* num-negate-blocks block-score)))

(define (adjust-image-with-path target-size img-path)
  (let* ([img (bitmap/file img-path)]
         [img-width (image-width img)]
         [img-height (image-height img)])
    ; Return an image whose width and height are equal to targe-size
    (scale/xy (/ target-size img-width)
              (/ target-size img-height) img)))
```

DSLs in Racket

tailor language-specific behavior

```
#lang clickomania

(define (create-background-image bsize bg-img-path img-path)
  (if (<= 400 bsize) {
        (define back-img (adjust-image-with-path bsize bg-img-path))
        (define img (adjust-image-with-path (* bsize 0.8) img-path))
        (define rotated-img (rotate -5 bimage))
        (underlay back-img rotated-img)
      }
      (adjust-image-with-path bsize bg-img-path)))

(define sunset-cloud (create-background-image 500 "./sunset.png" "./cloud.png"))
(set-background-image sunset-cloud)
(run 500 5 #:theme "candy" #:num-colors 4 #:num-clicks 10)
```


#lang clickomania
(define (create-background-image bsize bg-img-path img-path)
(if (<= 400 bsize) {
(define back-img (adjust-image-with-path bsize bg-img-path))
(define img (adjust-image-with-path (* bsize 0.8) img-path))
(define rotated-img (rotate -5 bimage))
(underlay back-img rotated-img)
}
(adjust-image-with-path bsize bg-img-path)))
(define sunset-cloud (create-background-image 500 "./sunset.png" "./cloud.png"))
(set-background-image sunset-cloud)
(run 500 5 #:theme "candy" #:num-colors 4 #:num-clicks 10)

```
#lang clickomania

(define (create-background-image bsize bg-img-path img-path)
  (if (<= 400 bsize) {
        (define back-img (adjust-image-with-path bsize bg-img-path))
        (define img (adjust-image-with-path (* bsize 0.8) img-path))
        (define rotated-img (rotate -5 bimage))
        (underlay back-img rotated-img)
      }
      (adjust-image-with-path bsize bg-img-path)))

(define sunset-cloud (create-background-image 500 "./sunset.png" "./cloud.png"))
(set-background-image sunset-cloud)
(run 500 5 #:theme "candy" #:num-colors 4 #:num-clicks 10)
```

```
#lang clickomania

(define (create-background-image bsize bg-img-path img
  (if (<= 400 bsize) {
        (define back-img (adjust-image-with-path bsize
        (define img (adjust-image-with-path (* bsize 0
        (define rotated-img (rotate -5 bimage))
        (underlay back-img rotated-img)
      }
      (adjust-image-with-path bsize bg-img-path)))

(define sunset-cloud (create-background-image 500 "./s
(set-background-image sunset-cloud)
(run 500 5 #:theme "candy" #:num-colors 4 #:num-clicks
```

```
#lang clickomania
(provide run
         create-background-image)

(define (game-info)
  (game-title "Tame Same Game")
  (game-author "Racketeer")
  (values (game-title) (game-author)))
```

```
#lang clickomania

(define (create-background-image bsize bg-img-path img
  (if (<= 400 bsize) {
        (define back-img (adjust-image-with-path bsize
        (define img (adjust-image-with-path (* bsize 0
        (define rotated-img (rotate -5 bimage))
        (underlay back-img rotated-img)
      }
      (adjust-image-with-path bsize bg-img-path)))

(define sunset-cloud (create-background-image 500 "./s
(set-background-image sunset-cloud)
(run 500 5 #:theme "candy" #:num-colors 4 #:num-clicks
```

```
#lang clickomania
(provide run
         create-background-image)

(define (game-info)
  (game-title "Tame Same Game")
  (game-author "Racketeer")
  (values (game-title) (game-author)))
```


#lang clickomania
(define (create-background-image bsize bg-img-path
(if (<= 400 bsize) {
(define back-img (adjust-image-with-path bsize
(define img (adjust-image-with-path (* bsize
(define rotated-img (rotate -5 bimage))
(underlay back-img rotated-img)
}
(adjust-image-with-path bsize bg-img-path)))
(define sunset-cloud (create-background-image 500 "./
(set-background-image sunset-cloud)
(run 500 5 #:theme "cand
#lang clickomania
(provide run
create-background-image)
(define (game-info)
(game-title "Tame Sa e")
-author)))
BANNED
#lang clickomania
#;(define (game-info)
(game-title "Tame Same Game")
(game-author "Racketeer")
(values (game-title) (game-author)))
Welcome to DrRacket, version 8.8.0.7--2023-10-23(-/f) [cs].
Language: Determine language from source; memory limit: 4096 MB.
provide: provided identifier is not defined or required in: game-info
>

client.rkt

```
#lang racket
(require racket/gui/base
         "config.rkt")
(provide author)
(define-values (title author) (game-info))
(define bg-frame
  (new frame%
       [label (format "~a's Window" author)]))

; Generate a game upon clicking a button
(define (on-button-click b e)
  (set-background-image
    (create-background-image 1000
                "./dirt.png" "./tree.png"))
  (run 1000 10 #:theme "gem"
               #:num-colors 5 #:num-clicks 20))

(new button% [parent bg-frame]
  [label (format "Play ~a!" title)]
  [callback
    (lambda (b e) (on-button-click b e))])
(send bg-frame show #t)
```

config.rkt

```
#lang clickomania
(provide run
         create-background-image)

(define (game-info)
  (game-title "Tame Same Game")
  (game-author "Racketeer")
  (values (game-title) (game-author)))
```

client.rkt

```
#lang racket
(require racket/gui/base
         "config.rkt")
(provide author)
(define-values (title author) (game-info))
(define bg-frame
  (new frame%
       [label (format "~a's Window" author)]))

; Generate a game upon clicking a button
(define (on-button-click b e)
  (set-background-image
    (create-background-image 1000
                "./dirt.png" "./tree.png"))
  (run 1000 10 #:theme "gem"
               #:num-colors 5 #:num-clicks 20))

(new button% [parent bg-frame]
  [label (format "Play ~a!" title)]
  [callback
    (lambda (b e) (on-button-click b e))])
(send bg-frame show #t)
```

config.rkt

```
#lang clickomania
(provide run
         create-background-image)

(define (game-info)
  (game-title "Tame Same Game")
  (game-author "Racketeer")
  (values (game-title) (game-author)))
```

client.rkt

```
#lang racket
(require racket/gui/base
         "config.rkt")
(provide author)
(define-values (title author) (game-info))
(define bg-frame
  (new frame%
       [label (format "~a's Window" author)]))

; Generate a game upon clicking a button
(define (on-button-click b e)
  (set-background-image
    (create-background-image 1000
                "./dirt.png" "./tree.png"))
  (run 1000 10 #:theme "gem"
               #:num-colors 5 #:num-clicks 20))

(new button% [parent bg-frame]
  [label (format "Play ~a!" title)]
  [callback
    (lambda (b e) (on-button-click b e))])
(send bg-frame show #t)
```

config.rkt

```
#lang clickomania
(provide run
         create-background-image)

(define (game-info)
  (game-title "Tame Same Game")
  (game-author "Racketeer")
  (values (game-title) (game-author)))
```

```
#lang clickomania

(define (create-background-image bsize bg-img-path img-path)
  (if (<= 400 bsize) {
        (define back-img (adjust-image-with-path bsize bg-img-path))
        (define img (adjust-image-with-path (* bsize 0.8) img-path))
        (define rotated-img (rotate -5 bimage))
```

client.rkt

```
#lang racket
(require racket/gui/base
         "config.rkt")
(provide author)
(define-values (title author) (game-info))
(define bg-frame
  (new frame%
       [label (format "~a's Window" author)]))

; Generate a game upon clicking a button
(define (on-button-click b e)
  (set-background-image
    (create-background-image 1000
                "./dirt.png" "./tree.png"))
  (run 1000 10 #:theme "gem"
               #:num-colors 5 #:num-clicks 2

(new button% [parent bg-frame]
  [label (format "Play ~a!" title)]
  [callback
    (lambda (b e) (on-button-click b e))])
(send bg-frame show #t)
```

```
#lang clickomania
(provide run
         create-background-image)

(define (game-info)
  (game-title "Tame Same Game")
  (game-author "Racketeer")
  (values (game-title) (game-author)))
```

mac.rkt

```
#lang clickomania
(provide adjust-image-with-path-macro)
; 'game-info' defined somewhere here
(define-syntax (adjust-image-with-path-macro
    stx)
  (syntax-parse stx
    [(_ size:number path:string)
     #'(adjust-image-with-path size path)]))
```

```
#lang clickomania

(define (create-background-image bsize bg-img-path img-path)
  (if (<= 400 bsize) {
        (define back-img (adjust-image-with-path bsize bg-img-path))
        (define img (adjust-image-with-path (* bsize 0.8) img-path))
        (define rotated-img (rotate -5 bimage))
```

client.rkt

```
#lang racket
(require racket/gui/base
         "config.rkt")
(provide author)
(define-values (title author) (game-info))
(define bg-frame
  (new frame%
       [label (format "~a's Window" author)]))

; Generate a game upon clicking a button
(define (on-button-click b e)
  (set-background-image
    (create-background-image 1000
                "./dirt.png" "./tree.png"))
  (run 1000 10 #:theme "gem"
               #:num-colors 5 #:num-clicks 2

(new button% [parent bg-frame]
  [label (format "Play ~a!" title)]
  [callback
    (lambda (b e) (on-button-click b e))])
(send bg-frame show #t)
```

```
#lang clickomania
(provide run
         create-background-image)

(define (game-info)
  (game-title "Tame Same Game")
  (game-author "Racketeer")
  (values (game-title) (game-author)))
```

mac.rkt

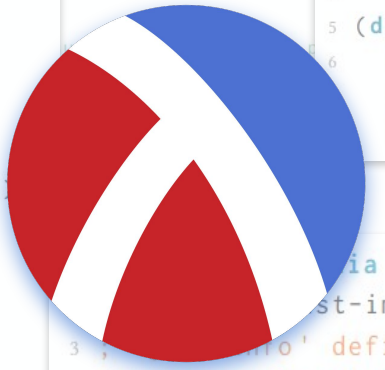
```
#lang clickomania
(provide adjust-image-with-path-macro)
; 'game-info' defined somewhere here
(define-syntax (adjust-image-with-path-macro
    stx)
  (syntax-parse stx
    [(_ size:number path:string)
     #'(adjust-image-with-path size path)]))
```

```
#lang clickomania

(define (create-background-image bsize bg-img-path img-path)
  (if (<= 400 bsize) {
        (define back-img (adjust-image-with-path bsize bg-img-path))
        (define img (adjust-image-with-path (* bsize 0.8) img-path))
        (define rotated-img (rotate -5 bimage))
```

client.rkt

```
#lang racket
(require racket/gui/base
         "config.rkt")
(provide author)
(define-values (title author) (game-info))
(define bg-frame
  (new frame%
       [label (format "~a's Window" author)]

; Generate a game upon clicking a button
(define (on-button-click b e)
  (set-background-image
    (create-background-image 1000
                "./dirt.png" "./tree.png"))
  (run 1000 10 #:theme "gem"
               #:num-colors 5 #:num-clicks 20))

(new button% [parent bg-frame]
  [label (format "Play ~a!" title)]
  [callback
    (lambda (b e) (on-button-click b e))])
(send bg-frame show #t)
```

```
#lang clickomania
(provide run
         create-background-image)

(define (game-info)
  (game-title "Tame Same Game")
  (game-author "Racketeer")
  (values (game-title) (game-author)))
```



mac.rkt

```
ia
st-image-with-path-macro)
; ...o' defined somewhere here
(define-syntax (adjust-image-with-path-macro
    stx)
  (syntax-parse stx
    [(_ size:number path:string)
     #'(adjust-image-with-path size path)]))
```

In the rest of the talk

- Language Design Intents
- Enabling Mechanisms
- Survey of DSLs in Racket

In the rest of the talk

- Language Design Intents
- Enabling Mechanisms
- Survey of DSLs in Racket

```
#lang rash

echo "Hello rash"
echo Show hidden files or a long listing:
ls (if (even? (random 2)) '-a '-l) | tac
```

```
#lang frog/config

(define/contract (init) (-> any)
  (current-title "My Blog"))
(define/contract (enhance-body xs)
  (-> (listof xexpr/c) (listof xexpr/c))
  (~> xs (syntax-highlight #:line-numbers? #t)))
(define/contract (clean) (-> any) (void))
```

```
#lang typed/racket

(define-type StrNum (Pairof String Number))

(: my-print-lst (-> (Listof StrNum) Void))
(define (my-print-lst lst)
  (for ([elem lst])
    (printf (car elem) (cdr elem))))
```

```
#lang rash

echo "Hello rash"
echo Show hidden files or a long listing:
ls (if (even? (random 2)) '-a '-l) | tac
```

```
#lang frog/config

(define/contract (init) (-> any)
  (current-title "My Blog"))
(define/contract (enhance-body xs)
  (-> (listof xexpr/c) (listof xexpr/c))
  (~> xs (syntax-highlight #:line-numbers? #t)))
(define/contract (clean) (-> any) (void))
```

```
#lang typed/racket

(define-type StrNum (Pairof String Number))

(: my-print-lst (-> (Listof StrNum) Void))
(define (my-print-lst lst)
  (for ([elem lst])
    (printf (car elem) (cdr elem))))
```

```
#lang rash

echo "Hello rash"
echo Show hidden files or a long listing:
ls (if (even? (random 2)) '-a '-l) | tac
```

```
#lang frog/config

(define/contract (init) (-> any)
  (current-title "My Blog"))
(define/contract (enhance-body xs)
  (-> (listof xexpr/c) (listof xexpr/c))
  (~> xs (syntax-highlight #:line-numbers? #t)))
(define/contract (clean) (-> any) (void))
```

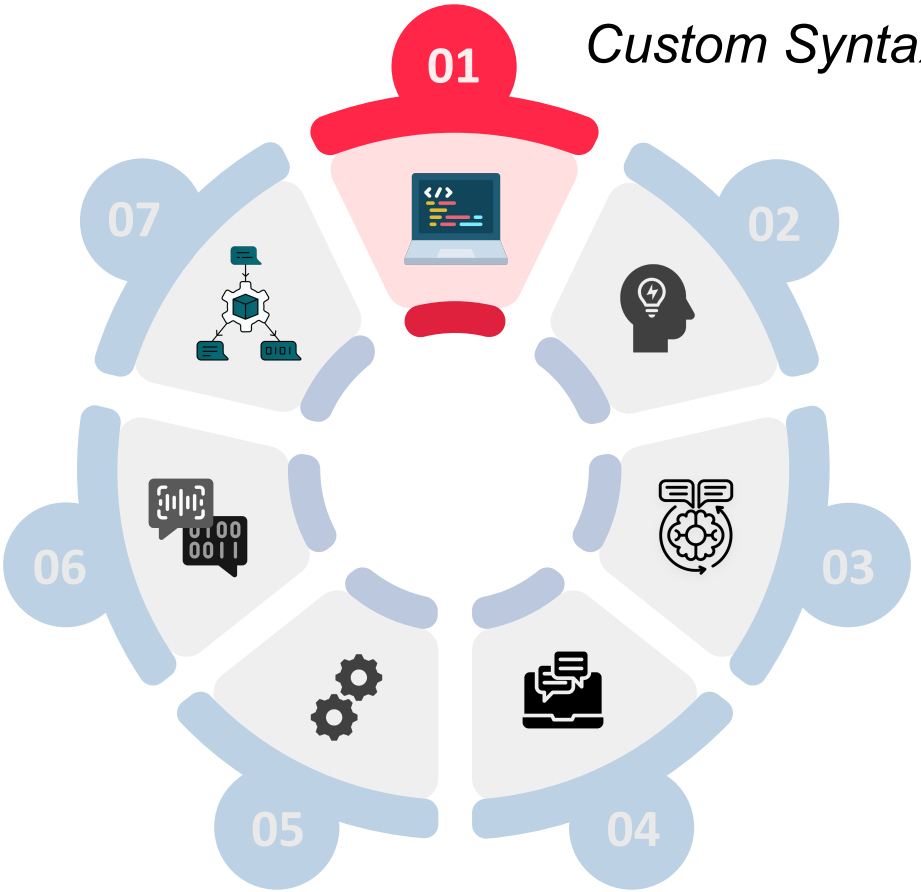
```
#lang typed/racket

(define-type StrNum (Pairof String Number))

(: my-print-lst (-> (Listof StrNum) Void))
(define (my-print-lst lst)
  (for ([elem lst])
    (printf (car elem) (cdr elem))))
```

```
#lang rash

echo "Hello rash"
echo Show hidden files or a long listing:
ls (if (even? (random 2)) '-a '-l) | tac
```

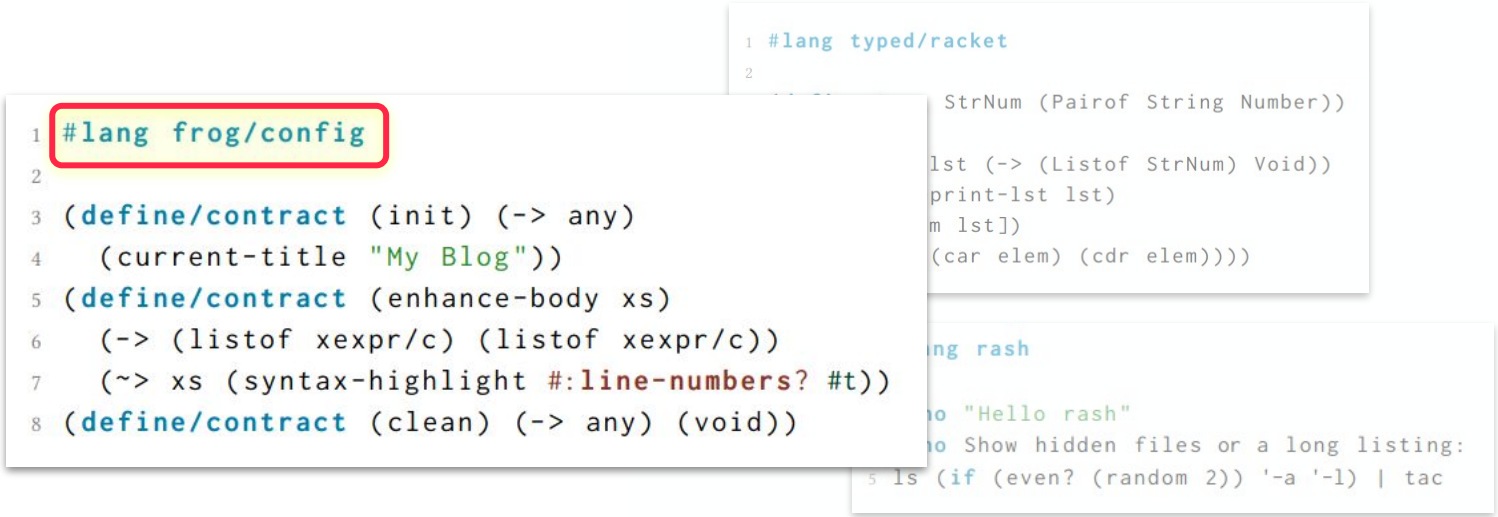
```
#lang frog/config

(define/contract (init) (-> any)
  (current-title "My Blog"))
(define/contract (enhance-body xs)
  (-> (listof xexpr/c) (listof xexpr/c))
  (~> xs (syntax-highlight #:line-numbers? #t)))
(define/contract (clean) (-> any) (void))
```

```
#lang typed/racket

(define-type StrNum (Pairof String Number))

(: my-print-lst (-> (Listof StrNum) Void))
(define (my-print-lst lst)
  (for ([elem lst])
    (printf (car elem) (cdr elem))))
```

Custom Syntax



Custom Syntax



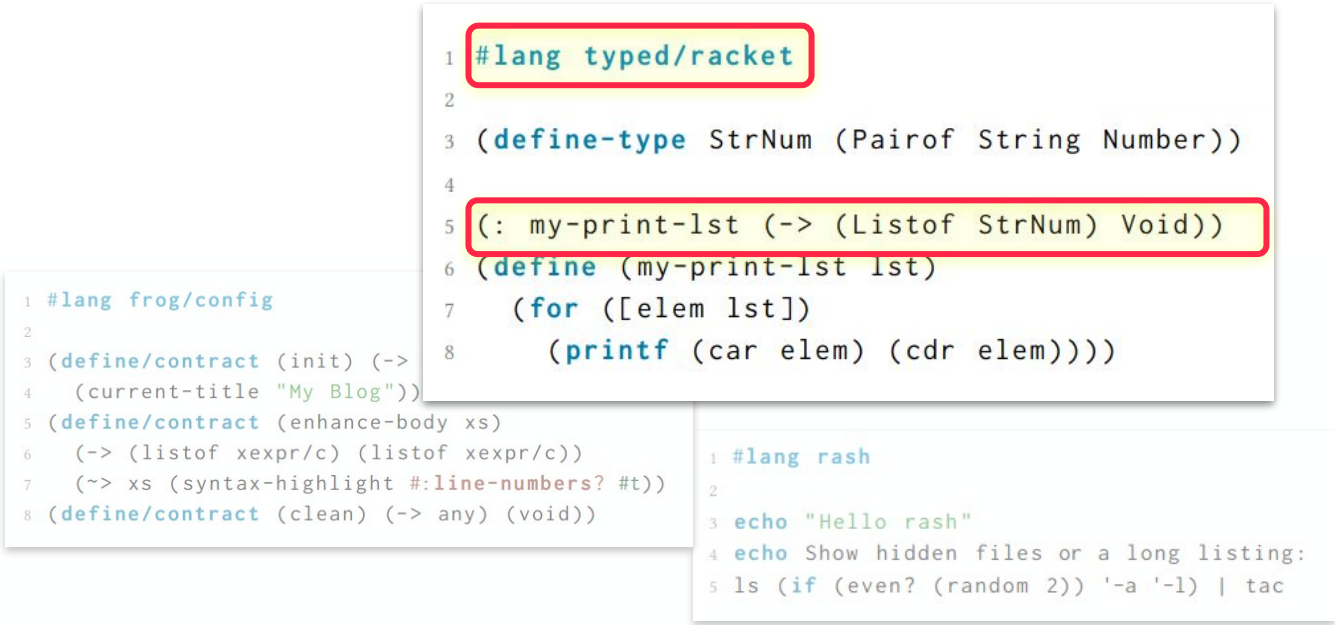
Custom Syntax

```
#lang frog/config

(define/contract (init) (-> any)
  (current-title "My Blog"))
(define/contract (enhance-body xs)
  (-> (listof xexpr/c) (listof xexpr/c))
  (~> xs (syntax-highlight #:line-numbers? #t))
(define/contract (clean) (-> any) (void))
```

```
#lang typed/racket

(define-type StrNum (Pairof String Number))

(: my-print-lst (-> (Listof StrNum) Void))
(define (my-print-lst lst)
  (for ([elem lst])
    (printf (car elem) (cdr elem))))
```

```
#lang rash

echo "Hello rash"
echo Show hidden files or a long listing:
ls (if (even? (random 2)) '-a '-l) | tac
```

Custom Syntax



```
#lang typed/racket

(define-type StrNum (Pairof String Number))

(: my-print-lst (-> (Listof StrNum) Void))
(define (my-print-lst lst)
  (for ([elem lst])
    (printf (car elem) (cdr elem))))
```

```
#lang frog/config

(define/contract (init) (->
  (current-title "My Blog"))
(define/contract (enhance-body xs)
  (-> (listof xexpr/c) (listof xexpr/c))
  (~> xs (syntax-highlight #:line-numbers? #t))
(define/contract (clean) (-> any) (void))
```

```
#lang rash

echo "Hello rash"
echo Show hidden files or a long listing:
ls (if (even? (random 2)) '-a '-l) | tac
```

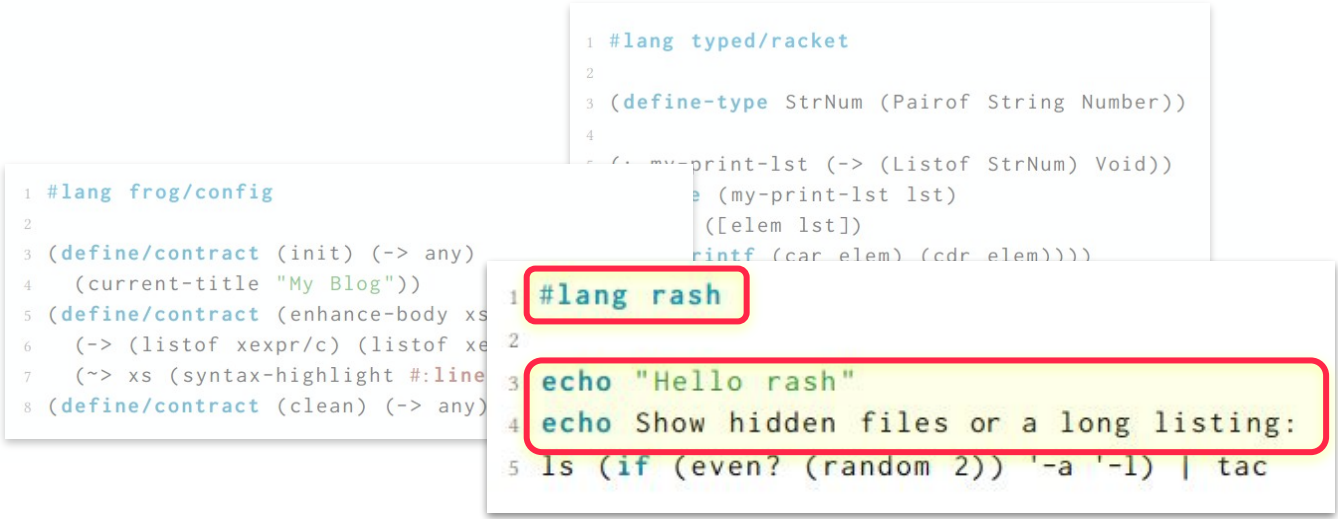
Custom Syntax

```
#lang frog/config

(define/contract (init) (-> any)
  (current-title "My Blog"))
(define/contract (enhance-body xs)
  (-> (listof xexpr/c) (listof xexpr/c))
  (~> xs (syntax-highlight #:line-numbers? #t))
(define/contract (clean) (-> any) (void))
```

```
#lang typed/racket

(define-type StrNum (Pairof String Number))

(: my-print-lst (-> (Listof StrNum) Void))
(define (my-print-lst lst)
  (for ([elem lst])
    (printf (car elem) (cdr elem))))
```

```
#lang rash

echo "Hello rash"
echo Show hidden files or a long listing:
ls (if (even? (random 2)) '-a '-l) | tac
```

Custom Syntax

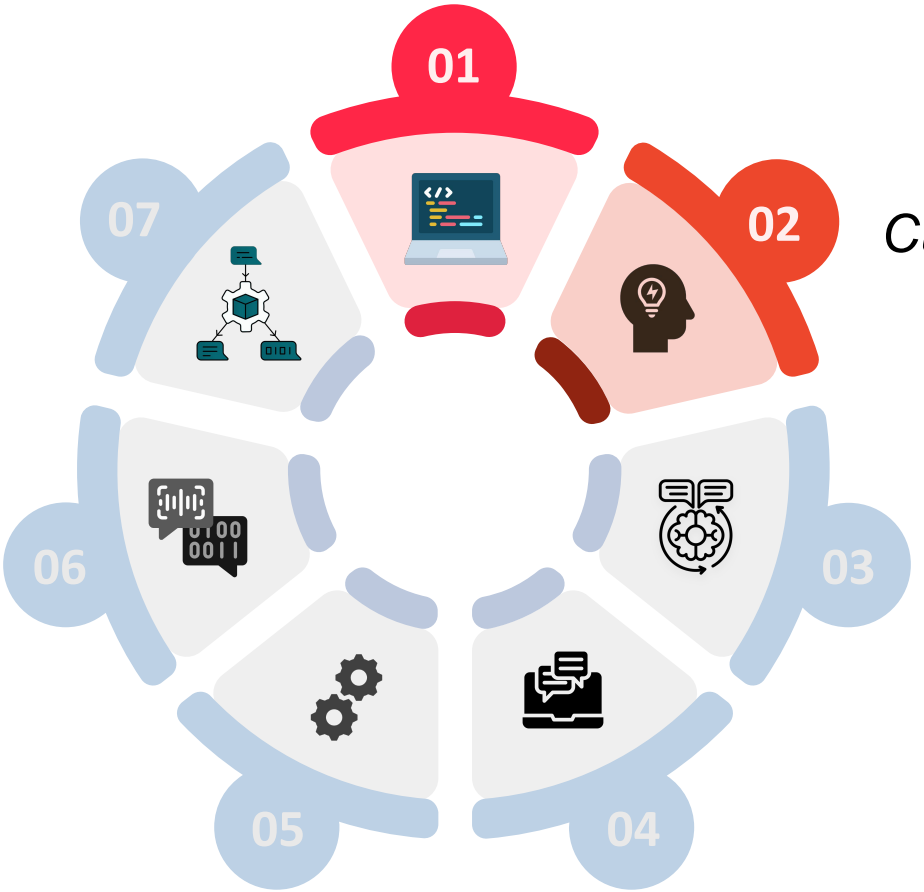


Custom Syntax

```
#lang frog/config

(define/contract (init) (-> any)
  (current-title "My Blog"))
(define/contract (enhance-body xs)
  (-> (listof xexpr/c) (listof xexpr/c))
  (~> xs (syntax-highlight #:line-numbers? #t))
(define/contract (clean) (-> any) (void))
```

```
#lang typed/racket

(define-type StrNum (Pairof String Number))

(: my-print-lst (-> (Listof StrNum) Void))
(define (my-print-lst lst)
  (for ([elem lst])
    (printf (car elem) (cdr elem))))
```

```
#lang rash

echo "Hello rash"
echo Show hidden files or a long listing:
ls (if (even? (random 2)) '-a '-l) | tac
```



Custom Semantics

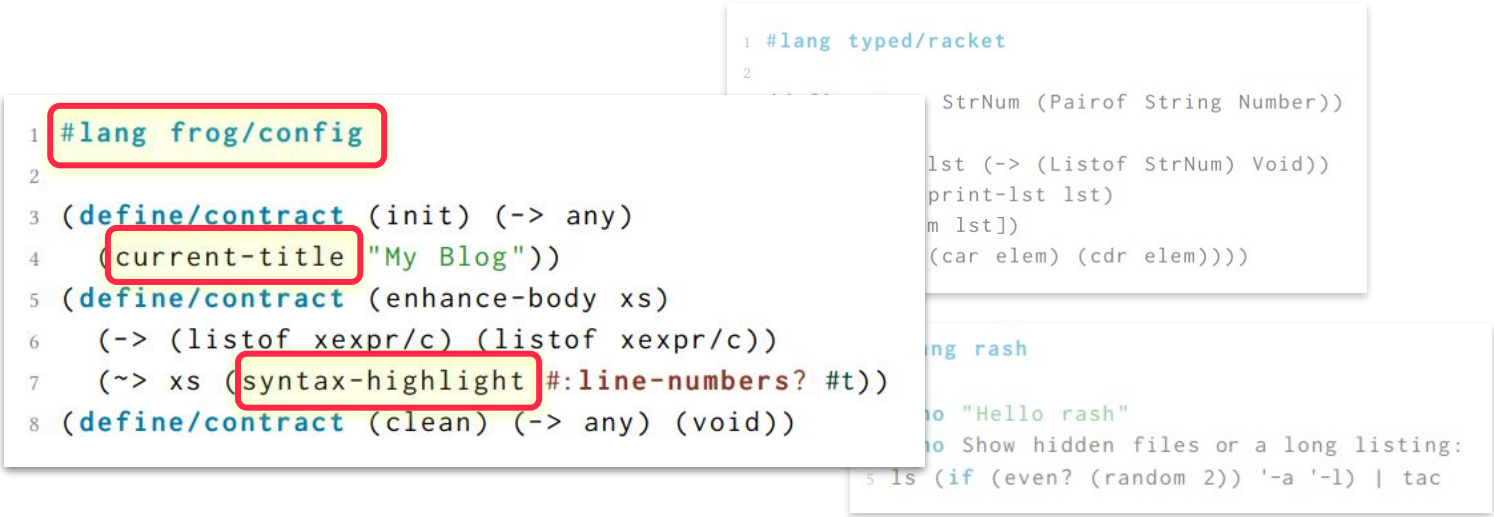
Custom Semantics

```
#lang frog/config

(define/contract (init) (-> any)
  (current-title "My Blog"))
(define/contract (enhance-body xs)
  (-> (listof xexpr/c) (listof xexpr/c))
  (~> xs (syntax-highlight #:line-numbers? #t))
(define/contract (clean) (-> any) (void))
```

```
#lang typed/racket

(define-type StrNum (Pairof String Number))

(: my-print-lst (-> (Listof StrNum) Void))
(define (my-print-lst lst)
  (for ([elem lst])
    (printf (car elem) (cdr elem))))
```

```
#lang rash

echo "Hello rash"
echo Show hidden files or a long listing:
ls (if (even? (random 2)) '-a '-l) | tac
```

Custom Semantics



Custom Semantics

```
#lang frog/config

(define/contract (init) (-> any)
  (current-title "My Blog"))
(define/contract (enhance-body xs)
  (-> (listof xexpr/c) (listof xexpr/c))
  (~> xs (syntax-highlight #:line-numbers? #t))
(define/contract (clean) (-> any) (void))
```

```
#lang typed/racket

(define-type StrNum (Pairof String Number))

(: my-print-lst (-> (Listof StrNum) Void))
(define (my-print-lst lst)
  (for ([elem lst])
    (printf (car elem) (cdr elem))))
```

```
#lang rash

echo "Hello rash"
echo Show hidden files or a long listing:
ls (if (even? (random 2)) '-a '-l) | tac
```

Custom Semantics

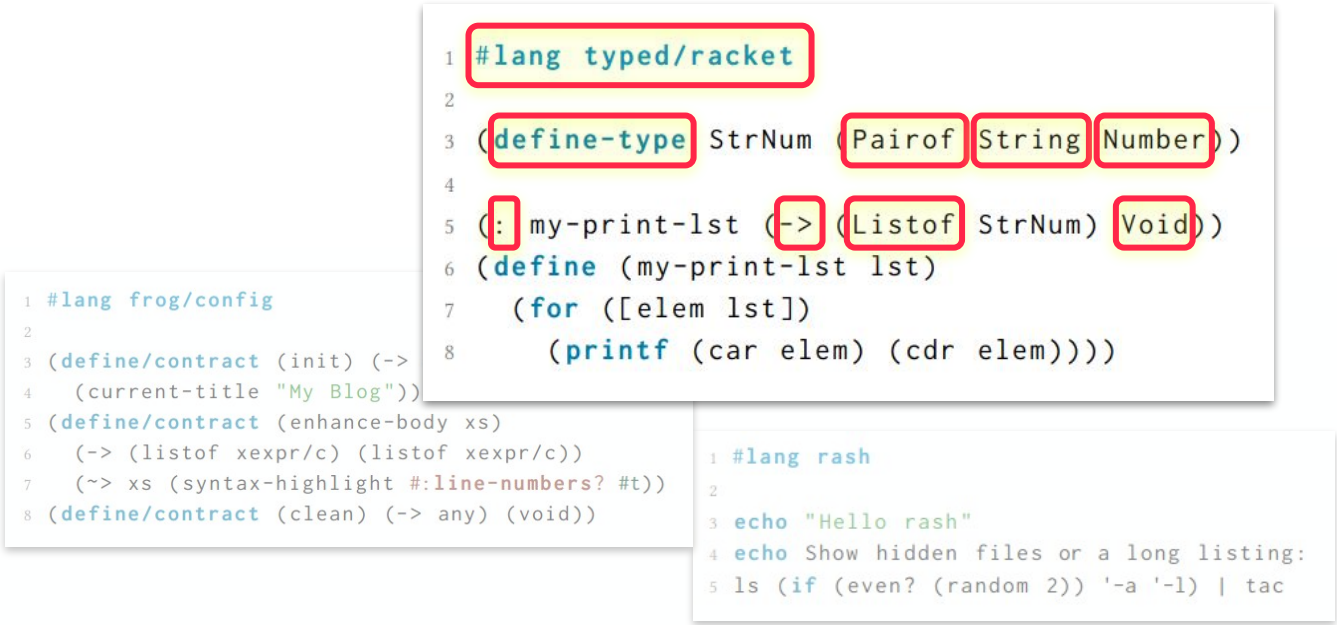
Custom Semantics

```
#lang frog/config

(define/contract (init) (-> any)
  (current-title "My Blog"))
(define/contract (enhance-body xs)
  (-> (listof xexpr/c) (listof xexpr/c))
  (~> xs (syntax-highlight #:line-numbers? #t))
(define/contract (clean) (-> any) (void))
```

```
#lang typed/racket

(define-type StrNum (Pairof String Number))

(: my-print-lst (-> (Listof StrNum) Void))
(define (my-print-lst lst)
  (for ([elem lst])
    (printf (car elem) (cdr elem))))
```

```
#lang rash

echo "Hello rash"
echo Show hidden files or a long listing:
ls (if (even? (random 2)) '-a '-l) | tac
```

Custom Semantics



Custom Semantics

```
#lang typed/racket

(define-type StrNum (Pairof String Number))

(: my-print-lst (-> (Listof StrNum) Void))
(define (my-print-lst lst)
  (for ([elem lst])
    (printf (car elem) (cdr elem))))
```

```
#lang frog/config

(define/contract (init) (->
  (current-title "My Blog"))
(define/contract (enhance-body xs)
  (-> (listof xexpr/c) (listof xexpr/c))
  (~> xs (syntax-highlight #:line-numbers? #t))
(define/contract (clean) (-> any) (void))
```

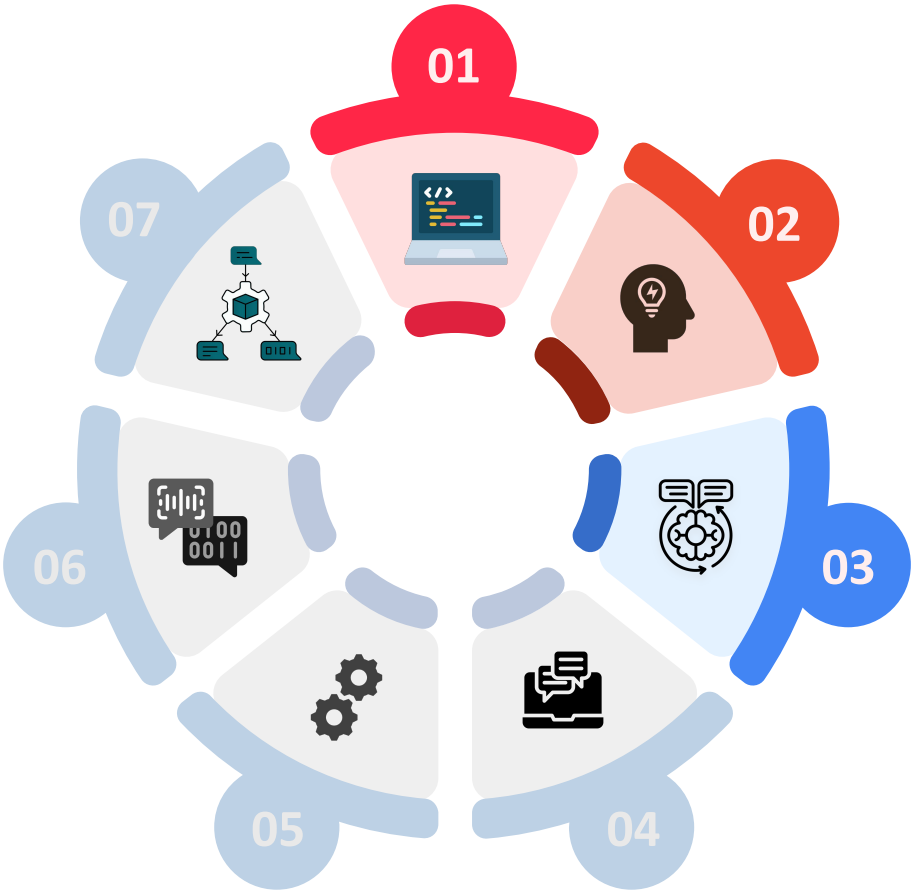
```
#lang rash

echo "Hello rash"
echo Show hidden files or a long listing:
ls (if (even? (random 2)) '-a '-l) | tac
```

Custom Semantics

```
#lang frog/config

(define/contract (init) (-> any)
  (current-title "My Blog"))
(define/contract (enhance-body xs)
  (-> (listof xexpr/c) (listof xexpr/c))
  (~> xs (syntax-highlight #:line-numbers? #t))
(define/contract (clean) (-> any) (void))
```

```
#lang typed/racket

(define-type StrNum (Pairof String Number))

(: my-print-lst (-> (Listof StrNum) Void))
(define (my-print-lst lst)
  (for ([elem lst])
    (printf (car elem) (cdr elem))))
```

```
#lang rash

echo "Hello rash"
echo Show hidden files or a long listing:
ls (if (even? (random 2)) '-a '-l) | tac
```



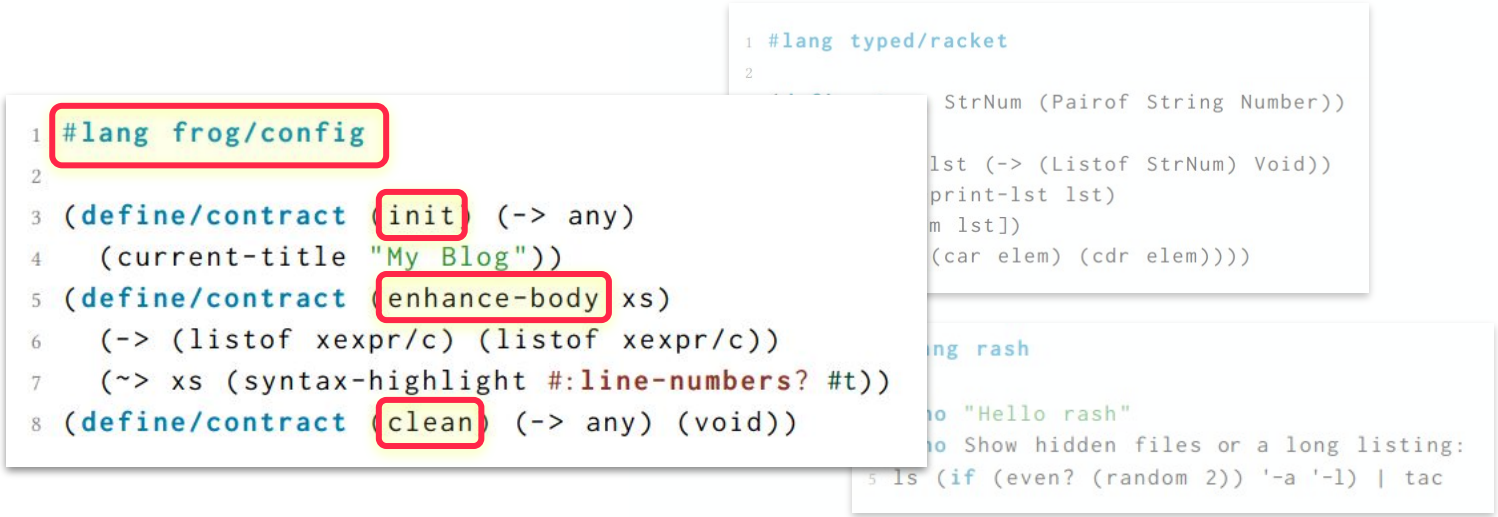
Import/Export Control

```
#lang frog/config

(define/contract (init) (-> any)
  (current-title "My Blog"))
(define/contract (enhance-body xs)
  (-> (listof xexpr/c) (listof xexpr/c))
  (~> xs (syntax-highlight #:line-numbers? #t))
(define/contract (clean) (-> any) (void))
```

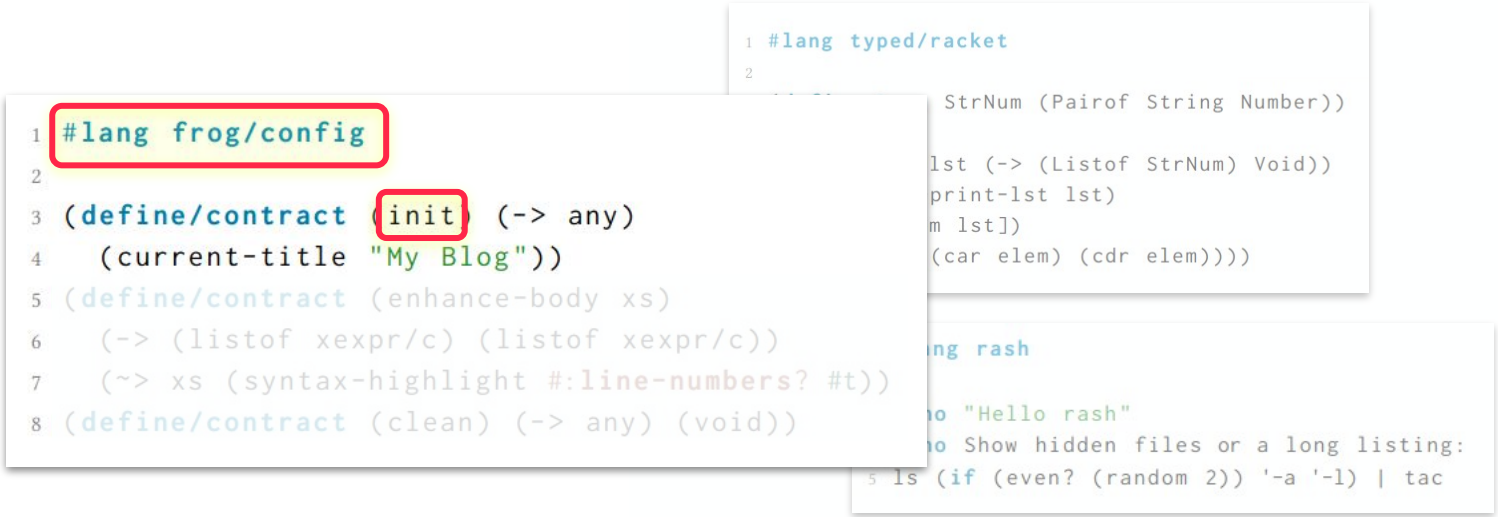
```
#lang typed/racket

(define-type StrNum (Pairof String Number))

(: my-print-lst (-> (Listof StrNum) Void))
(define (my-print-lst lst)
  (for ([elem lst])
    (printf (car elem) (cdr elem))))
```

```
#lang rash

echo "Hello rash"
echo Show hidden files or a long listing:
ls (if (even? (random 2)) '-a '-l) | tac
```

Import/Export Control



Import/Export Control



Import/Export Control

typed-fun.rkt

```
#lang typed/racket

(define-type StrNum (Pairof String Number))

(: my-print-lst (-> (Listof StrNum) Void))
(define (my-print-lst lst)
  (for ([elem lst])
    (printf (car elem) (cdr elem))))
```

```
#lang frog/config

(define/contract (init) (-> any)
  (current-title "My Blog"))
(define/contract (enhance-body xs)
  (-> (listof xexpr/c) (listof xexpr/c))
  (~> xs (syntax-highlight #:line-numbers? #t)))
(define/contract (clean) (-> any) (void))
```

```
#lang rash

echo "Hello rash"
echo Show hidden files or a long listing:
ls (if (even? (random 2)) '-a '-l) | tac
```

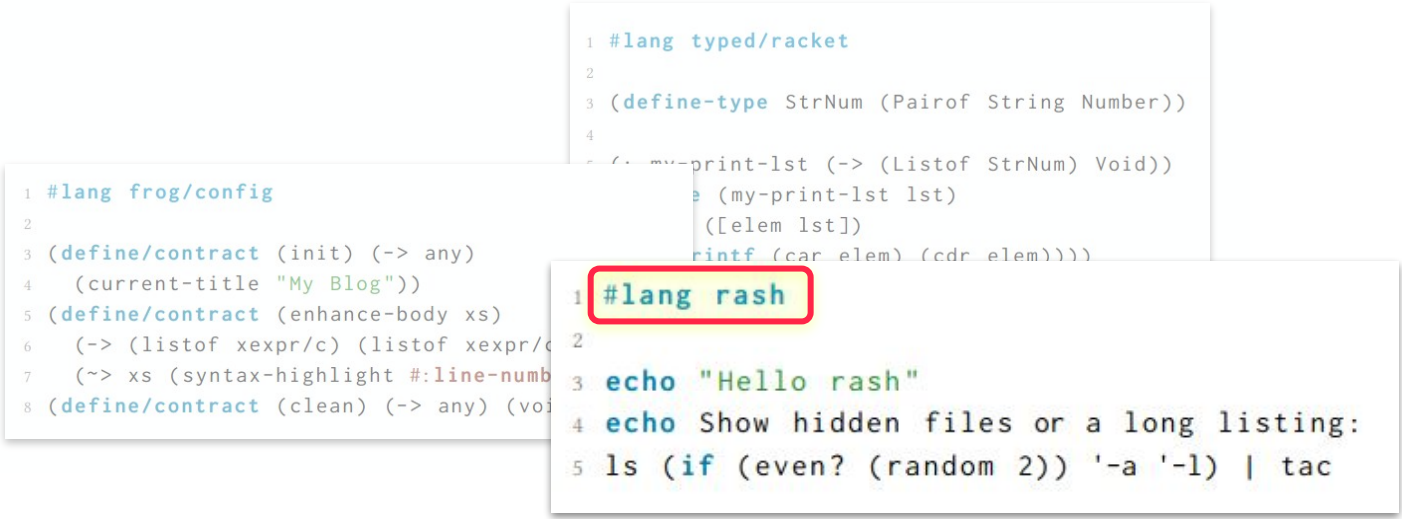
```
#lang typed/racket

(define-type StrNum (Pairof String Number))

(: my-print-lst (-> (Listof StrNum) Void))
(define (my-print-lst lst)
  (for ([elem lst])
    (printf (car elem) (cdr elem))))
```

typed-fun.rkt

```
#lang frog/config
```

```
#lang typed/racket
(require "typed-fun.rkt")
(require/typed racket
               [string-append (-> String String String)]
               [for-each (-> (-> StrNum Void) (Listof StrNum) Void)])

(: my-println-lst (-> (Listof StrNum) Void))
(define (my-println-lst lst)
  (for-each (lambda ([elem : StrNum])
              (let ([new-str (string-append (car elem) " ~a\n")]
                    [num (cdr elem)])
                (printf new-str num)))
            lst))
```

```
files or a long listing:
ndom 2)) '-a '-l) | tac
```



```
#lang frog/config

(define/contract (init) (-> any)
  (current-title "My Blog"))
(define/contract (enhance-body xs)
  (-> (listof xexpr/c) (listof xexpr/c
  (~> xs (syntax-highlight #:line-numb
(define/contract (clean) (-> any) (voi
```

```
#lang typed/racket

(define-type StrNum (Pairof String Number))

(: my-print-lst (-> (Listof StrNum) Void))
(define (my-print-lst lst)
  (for ([elem lst])
    (printf (car elem) (cdr elem))))
```

```
#lang rash

echo "Hello rash"
echo Show hidden files or a long listing:
ls (if (even? (random 2)) '-a '-l) | tac
```

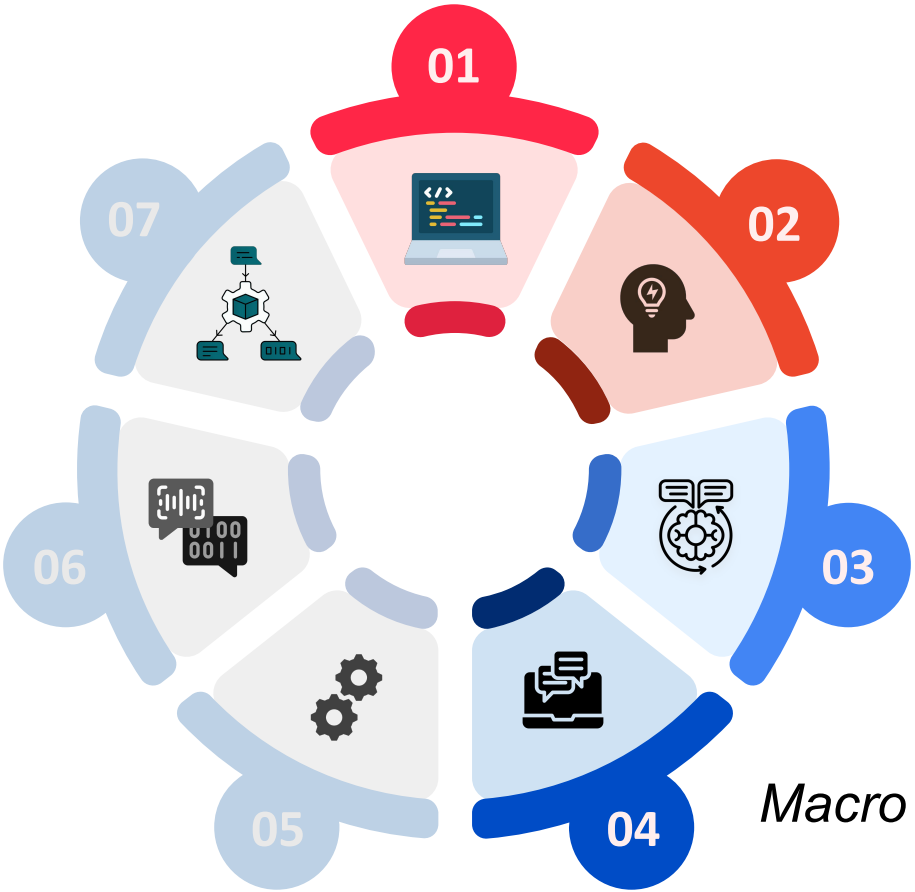

1 #lang rash
2 (require "./cake.rkt")
3
4 echo "Hello rash"
5 echo Show hidden files or give a long listing:
6 ls (if (even? (random 2)) '-a '-l) | tac
7 echo (print-cake 5)
8
total 172
.....
.-|||||-.
| |

#<void>
>
1 #lang typed/racket
2
3 (define-type StrNum (Pairof String Number))
4
(my-print-lst lst)
([elem lst])
1 #lang rash
2
3 echo "Hello rash"
4 echo Show hidden files or a long listing:
5 ls (if (even? (random 2)) '-a '-l) | tac

```
#lang frog/config

(define/contract (init) (-> any)
  (current-title "My Blog"))
(define/contract (enhance-body xs)
  (-> (listof xexpr/c) (listof xexpr/c))
  (~> xs (syntax-highlight #:line-numbers? #t))
(define/contract (clean) (-> any) (void))
```

```
#lang typed/racket

(define-type StrNum (Pairof String Number))

(: my-print-lst (-> (Listof StrNum) Void))
(define (my-print-lst lst)
  (for ([elem lst])
    (printf (car elem) (cdr elem))))
```

```
#lang rash

echo "Hello rash"
echo Show hidden files or a long listing:
ls (if (even? (random 2)) '-a '-l) | tac
```



Macro Support

Macro Support

```
#lang frog/config

(define/contract (init) (-> any)
  (current-title "My Blog"))
(define/contract (enhance-body xs)
  (-> (listof xexpr/c) (listof xexpr/c))
  (~> xs (syntax-highlight #:line-numbers? #t))
(define/contract (clean) (-> any) (void))
```

```
#lang typed/racket

(define-type StrNum (Pairof String Number))

(: my-print-lst (-> (Listof StrNum) Void))
(define (my-print-lst lst)
  (for ([elem lst])
    (printf (car elem) (cdr elem))))
```

```
#lang rash

echo "Hello rash"
echo Show hidden files or a long listing:
ls (if (even? (random 2)) '-a '-l) | tac
```

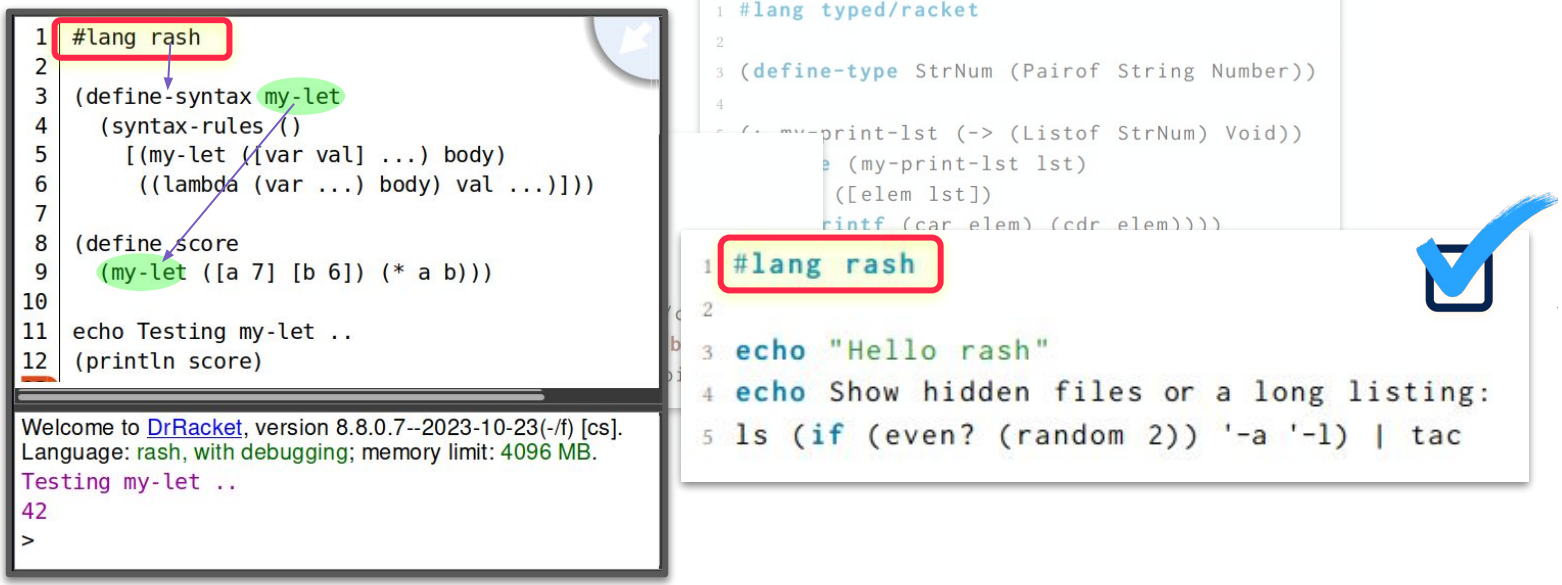
Macro Support

```
#lang frog/config

(define/contract (init) (-> any)
  (current-title "My Blog"))
(define/contract (enhance-body xs)
  (-> (listof xexpr/c) (listof xexpr/c
  (~> xs (syntax-highlight #:line-numb
(define/contract (clean) (-> any) (voi
```

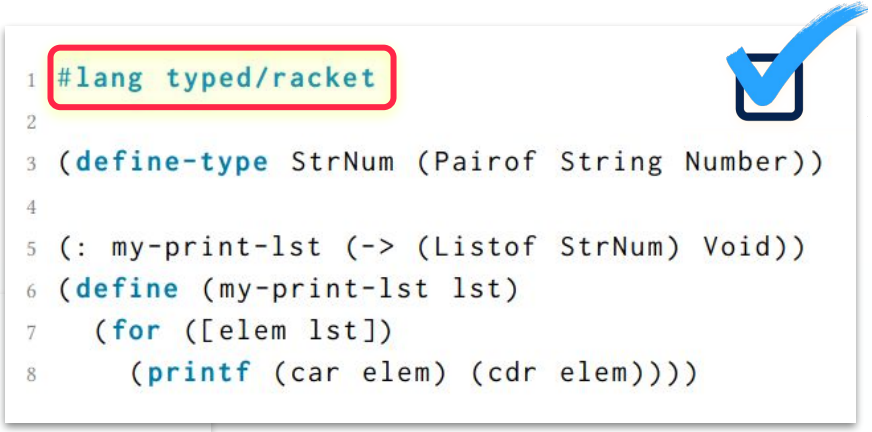
```
#lang typed/racket

(define-type StrNum (Pairof String Number))

(: my-print-lst (-> (Listof StrNum) Void))
(define (my-print-lst lst)
  (for ([elem lst])
    (printf (car elem) (cdr elem))))
```

```
#lang rash

echo "Hello rash"
echo Show hidden files or a long listing:
ls (if (even? (random 2)) '-a '-l) | tac
```

Macro Support



Macro Support

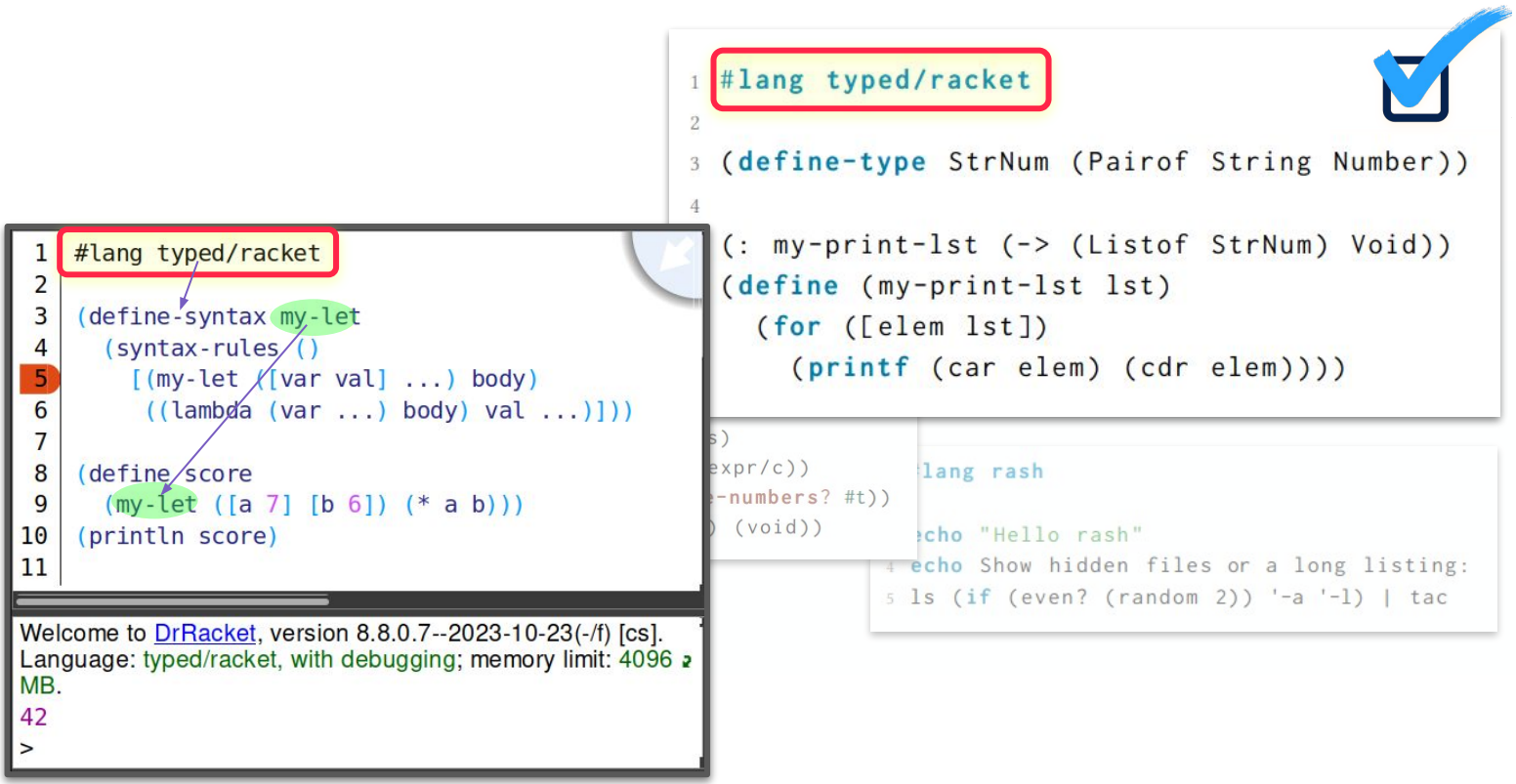


```
#lang typed/racket

(define-type StrNum (Pairof String Number))

(: my-print-lst (-> (Listof StrNum) Void))
(define (my-print-lst lst)
  (for ([elem lst])
    (printf (car elem) (cdr elem))))
```

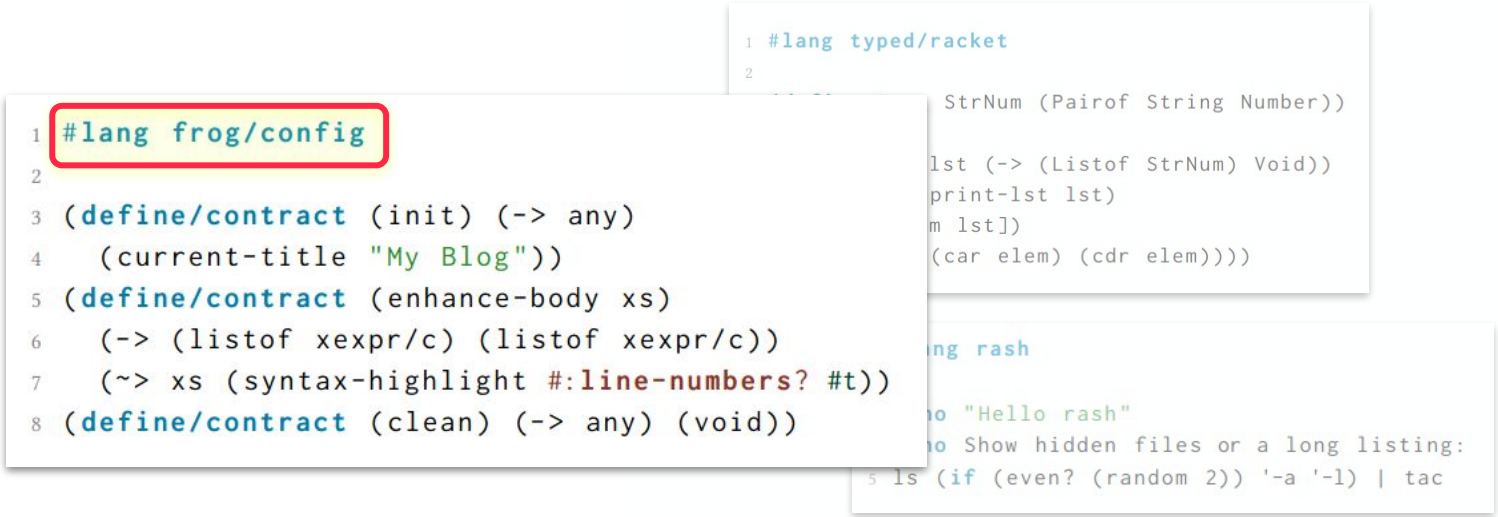
```
#lang frog/config

(define/contract (init) (-> any)
  (current-title "My Blog"))
(define/contract (enhance-body xs)
  (-> (listof xexpr/c) (listof xexpr/c))
  (~> xs (syntax-highlight #:line-numbers? #t))
(define/contract (clean) (-> any) (void))
```

```
#lang rash

echo "Hello rash"
echo Show hidden files or a long listing:
ls (if (even? (random 2)) '-a '-l) | tac
```

Macro Support



Macro Support



Macro Support

```
#lang frog/config

(define/contract (init) (-> any)
  (current-title "My Blog"))
(define/contract (enhance-body xs)
  (-> (listof xexpr/c) (listof xexpr/c))
  (~> xs (syntax-highlight #:line-numbers? #t)))
(define/contract (clean) (-> any) (void))
```

```
#lang typed/racket

 StrNum (Pairof String Number))

lst (-> (Listof StrNum) Void))
print-lst lst)
m lst])
(car elem) (cdr elem))))
```

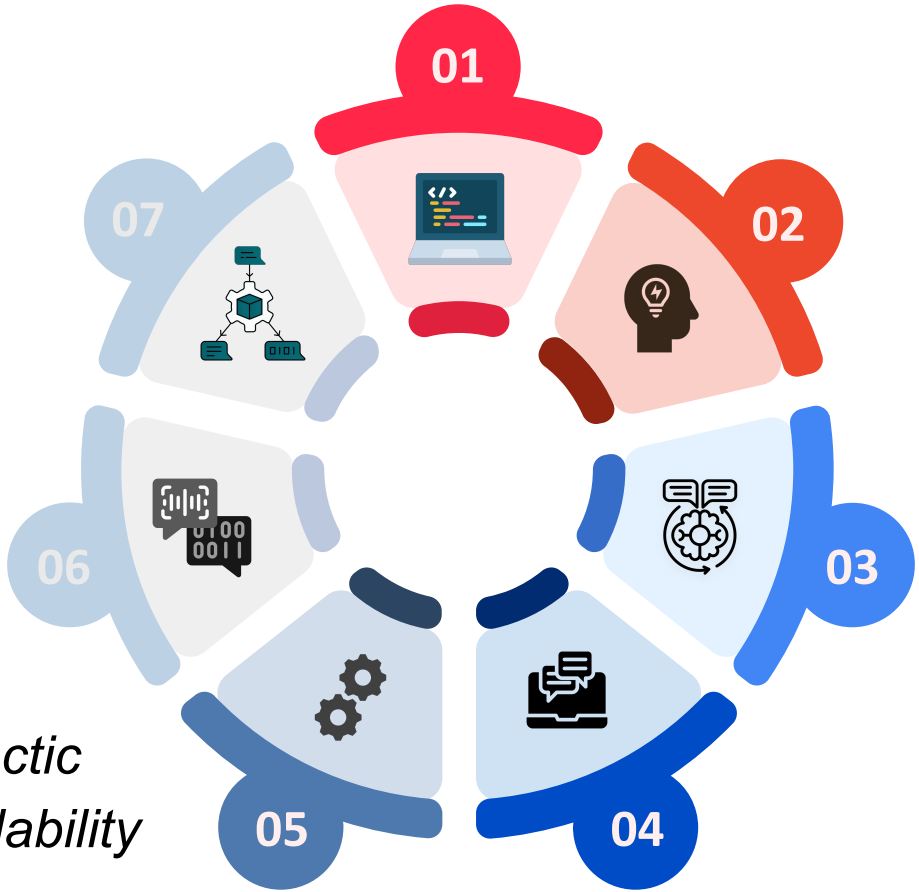
```
ng rash

o "Hello rash"
o Show hidden files or a long listing:
ls (if (even? (random 2)) '-a '-l) | tac
```

Macro Support

```
#lang frog/config

(define/contract (init) (-> any)
  (current-title "My Blog"))
(define/contract (enhance-body xs)
  (-> (listof xexpr/c) (listof xexpr/c))
  (~> xs (syntax-highlight #:line-numbers? #t))
(define/contract (clean) (-> any) (void))
```

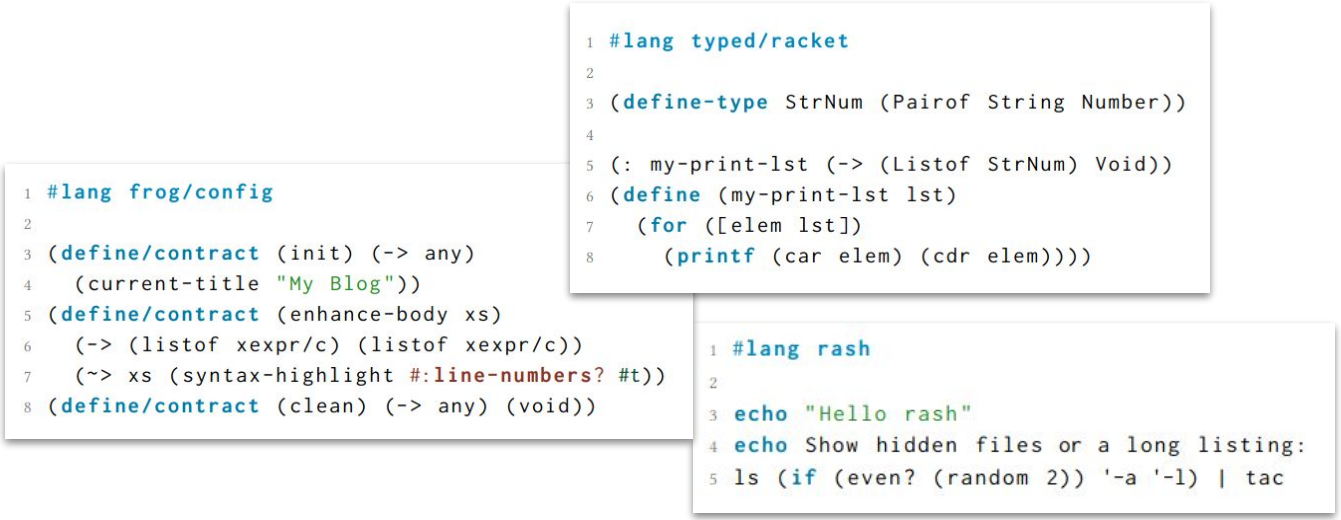
```
#lang typed/racket

(define-type StrNum (Pairof String Number))

(: my-print-lst (-> (Listof StrNum) Void))
(define (my-print-lst lst)
  (for ([elem lst])
    (printf (car elem) (cdr elem))))
```

```
#lang rash

echo "Hello rash"
echo Show hidden files or a long listing:
ls (if (even? (random 2)) '-a '-l) | tac
```



Syntactic
Embeddability

Syntactic Embeddability

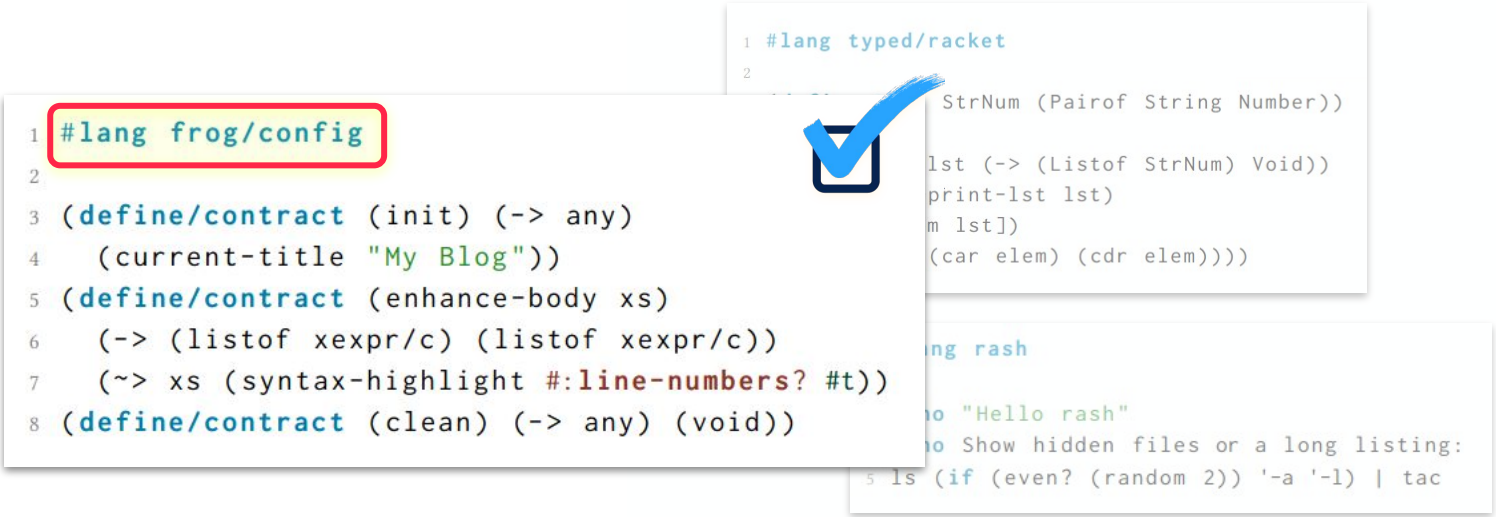


```
#lang frog/config

(define/contract (init) (-> any)
  (current-title "My Blog"))
(define/contract (enhance-body xs)
  (-> (listof xexpr/c) (listof xexpr/c))
  (~> xs (syntax-highlight #:line-numbers? #t))
(define/contract (clean) (-> any) (void))
```

```
#lang typed/racket

(define-type StrNum (Pairof String Number))

(: my-print-lst (-> (Listof StrNum) Void))
(define (my-print-lst lst)
  (for ([elem lst])
    (printf (car elem) (cdr elem))))
```

```
#lang rash

echo "Hello rash"
echo Show hidden files or a long listing:
ls (if (even? (random 2)) '-a '-l) | tac
```

Syntactic Embeddability

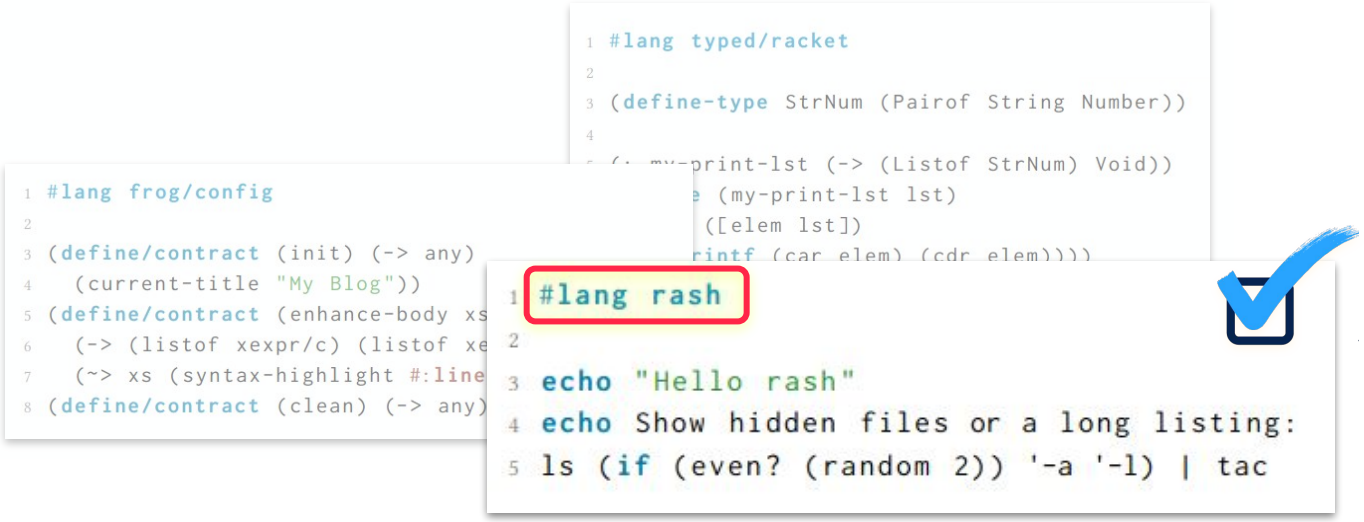


Syntactic Embeddability

```
#lang frog/config

(define/contract (init) (-> any)
  (current-title "My Blog"))
(define/contract (enhance-body xs)
  (-> (listof xexpr/c) (listof xexpr/c))
  (~> xs (syntax-highlight #:line-numbers? #t))
(define/contract (clean) (-> any) (void))
```

```
#lang typed/racket

(define-type StrNum (Pairof String Number))

(: my-print-lst (-> (Listof StrNum) Void))
(define (my-print-lst lst)
  (for ([elem lst])
    (printf (car elem) (cdr elem))))
```

```
#lang rash

echo "Hello rash"
echo Show hidden files or a long listing:
ls (if (even? (random 2)) '-a '-l) | tac
```


#lang typed/racket
(define-type StrNum (Pairof String Number))
#lang frog/config
(define/contract (init) (-> any)
(current-title "My Blog"))
(define/contract (enhance-body xs
(-> (listof xexpr/c) (listof xe
(define/contract (clean) (-> any)
#lang rash
echo "Hello rash"
echo Show hidden files or a long listing:
ls (if (even? (random 2)) '-a '-l) | tac

Syntactic Embeddability

```
#lang typed/racket

(define-type StrNum (Pairof String Number))

(: my-print-lst (-> (Listof StrNum) Void))
(define (my-print-lst lst)
  (for ([elem lst])
    (printf (car elem) (cdr elem))))
```

```
#lang frog/config

(define/contract (init) (-> any)
  (current-title "My Blog"))
(define/contract (enhance-body xs)
  (-> (listof xexpr/c) (listof xexpr/c))
  (~> xs (syntax-highlight #:line-numbers? #t))
(define/contract (clean) (-> any) (void))
```

```
#lang rash

echo "Hello rash"
echo Show hidden files or a long listing:
ls (if (even? (random 2)) '-a '-l) | tac
```

Syntactic Embeddability

```
#lang typed/racket

(define-type StrNum (Pairof String Number))

(: my-print-lst (-> (Listof StrNum) Void))
(define (my-print-lst lst)
  (for ([elem lst])
    (printf (car elem) (cdr elem))))
```

```
#lang frog/config

(define/contract (init) (-> any)
  (current-title "My Blog"))
(define/contract (enhance-body xs)
  (-> (listof xexpr/c) (listof xexpr/c))
  (~> xs (syntax-highlight #:line-numbers? #t))
(define/contract (clean) (-> any) (void))
```

```
#lang rash

echo "Hello rash"
echo Show hidden files or a long listing:
ls (if (even? (random 2)) '-a '-l) | tac
```

```
#lang typed/racket

(define-type StrNum (Pairof String Number))

(: my-print-lst (-> (Listof StrNum) Void))
(define (my-print-lst lst)
  (for ([elem lst])
    (printf (car elem) (cdr elem))))
```

```
#lang frog/config
```

```
#lang typed/racket
(require "typed-fun.rkt")
(require/typed racket
               [string-append (-> String String String)]
               [for-each (-> (-> StrNum Void) (Listof StrNum) Void)])

(: my-println-lst (-> (Listof StrNum) Void))
(define (my-println-lst lst)
  (for-each (lambda ([elem : StrNum])
              (let ([new-str (string-append (car elem) " ~a\n")]
                    [num (cdr elem)])
                (printf new-str num)))
            lst))
```

```
files or a long listing:
ndom 2)) '-a '-l) | tac
```

```
#lang typed/racket

(define-type StrNum (Pairof String Number))

(: my-print-lst (-> (Listof StrNum) Void))
(define (my-print-lst lst)
  (for ([elem lst])
    (printf (car elem) (cdr elem))))
```

```
#lang frog/config
```

```
files or a long listing:
andom 2)) '-a '-l) | tac
```

```
#lang typed/racket
(require "typed-fun.rkt")
(require/typed racket
               [string-append (-> String String String)]
               [for-each (-> (-> StrNum Void) (Listof StrNum) Void)])

(: my-println-lst (-> (Listof StrNum) Void))
(define (my-println-lst lst)
  (for-each (lambda ([elem : StrNum])
              (let ([new-str (string-append (car elem) " ~a\n")]
                    [num (cdr elem)])
                (printf new-str num)))
            lst))
```

```
#lang typed/racket

(define-type StrNum (Pairof String Number))

(: my-print-lst (-> (Listof StrNum) Void))
(define (my-print-lst lst)
  (for ([elem lst])
    (printf (car elem) (cdr elem))))
```

```
#lang frog/config
```

```
#lang typed/racket
(require "typed-fun.rkt")
(require/typed racket
               [string-append (-> String String String)]
               [for-each (-> (-> StrNum Void) (Listof StrNum) Void)])

(: my-println-lst (-> (Listof StrNum) Void))
(define (my-println-lst lst)
  (for-each (lambda ([elem : StrNum])
              (let ([new-str (string-append (car elem) " ~a\n")]
                    [num (cdr elem)])
                (printf new-str num)))
            lst))
```

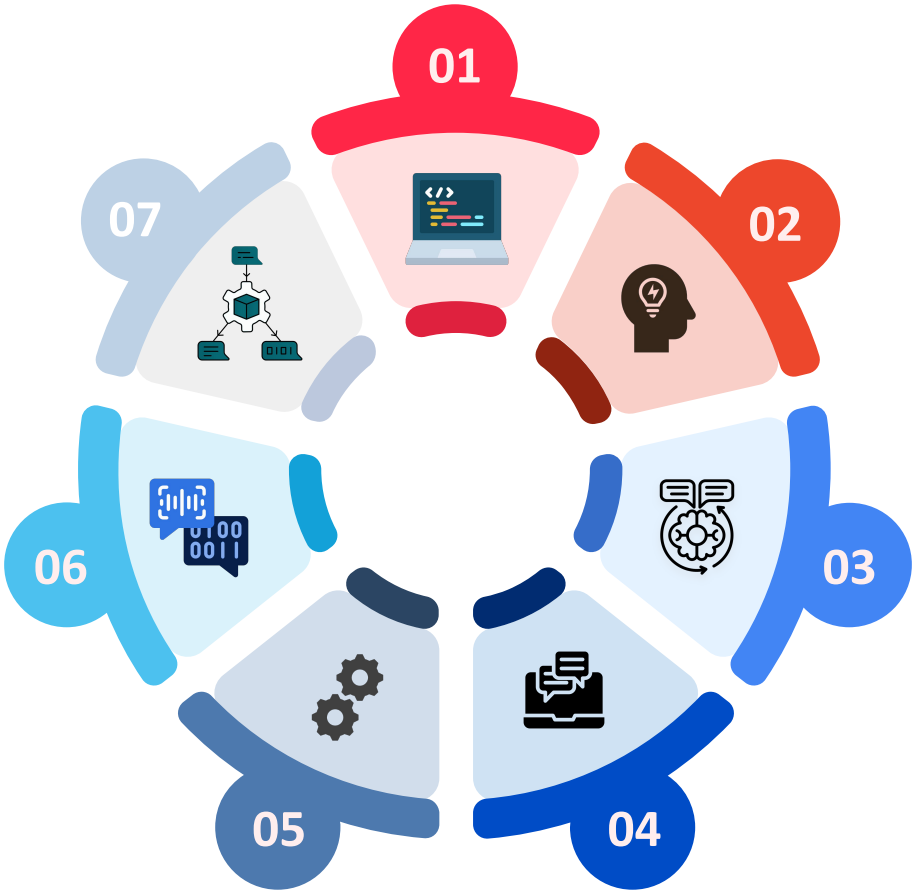
```
files or a long listing:
andom 2)) '-a '-l) | tac
```

```
#lang frog/config

(define/contract (init) (-> any)
  (current-title "My Blog"))
(define/contract (enhance-body xs)
  (-> (listof xexpr/c) (listof xexpr/c))
  (~> xs (syntax-highlight #:line-numbers? #t))
(define/contract (clean) (-> any) (void))
```

```
#lang typed/racket

(define-type StrNum (Pairof String Number))

(: my-print-lst (-> (Listof StrNum) Void))
(define (my-print-lst lst)
  (for ([elem lst])
    (printf (car elem) (cdr elem))))
```

```
#lang rash

echo "Hello rash"
echo Show hidden files or a long listing:
ls (if (even? (random 2)) '-a '-l) | tac
```

Interoperability



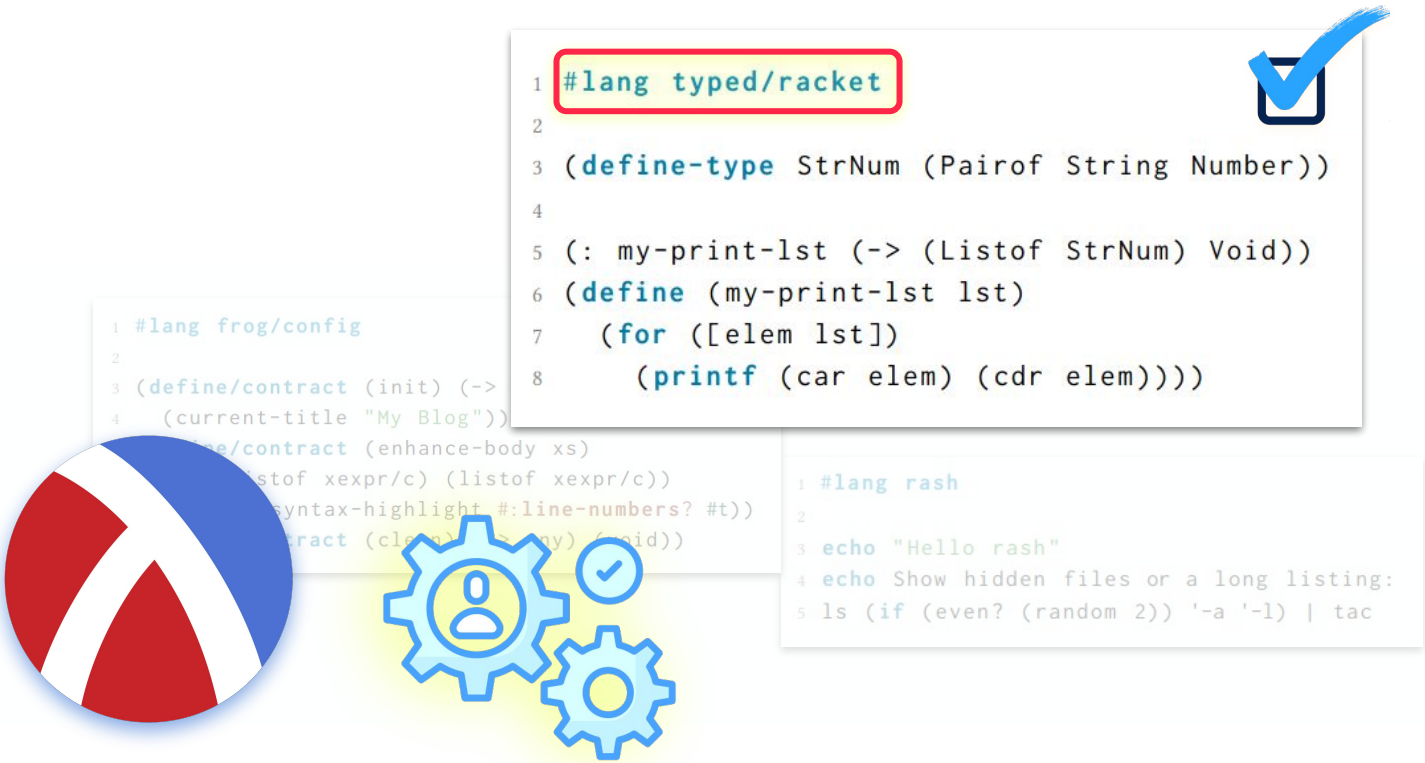
```
#lang frog/config

(define/contract (init) (-> any)
  (current-title "My Blog"))
(define/contract (enhance-body xs)
  (-> (listof xexpr/c) (listof xexpr/c))
  (~> xs (syntax-highlight #:line-numbers? #t))
(define/contract (clean) (-> any) (void))
```

```
#lang typed/racket

(define-type StrNum (Pairof String Number))

(: my-print-lst (-> (Listof StrNum) Void))
(define (my-print-lst lst)
  (for ([elem lst])
    (printf (car elem) (cdr elem))))
```

```
#lang rash

echo "Hello rash"
echo Show hidden files or a long listing:
ls (if (even? (random 2)) '-a '-l) | tac
```

```
#lang frog/config

(define/contract (init) (-> any)
  (current-title "My Blog"))
(define/contract (enhance-body xs)
  (-> (listof xexpr/c) (listof xexpr/c))
  (~> xs (syntax-highlight #:line-numbers? #t))
(define/contract (clean) (-> any) (void))
```

Interoperability



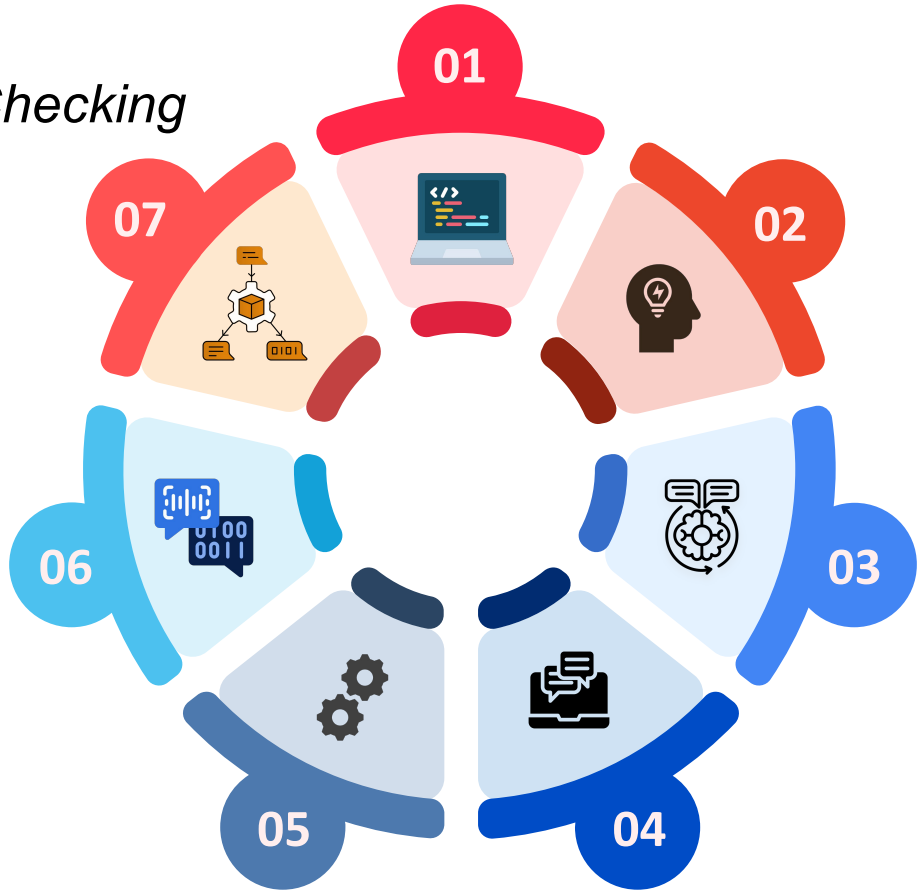

#lang typed/racket
(define-type StrNum (Pairof String Number))
(: my-print-lst (-> (Listof StrNum) Void))
(define (my-print-lst lst)
(for ([elem lst])
#lang rash
echo "Hello rash"
echo Show hidden files or a long listing:
ls (if (even? (random 2)) '-a '-l) | tac

```
#lang frog/config

(define/contract (init) (-> any)
  (current-title "My Blog"))
(define/contract (enhance-body xs)
  (-> (listof xexpr/c) (listof xexpr/c))
  (~> xs (syntax-highlight #:line-numbers? #t))
(define/contract (clean) (-> any) (void))
```

```
#lang typed/racket

(define-type StrNum (Pairof String Number))

(: my-print-lst (-> (Listof StrNum) Void))
(define (my-print-lst lst)
  (for ([elem lst])
    (printf (car elem) (cdr elem))))
```

```
#lang rash

echo "Hello rash"
echo Show hidden files or a long listing:
ls (if (even? (random 2)) '-a '-l) | tac
```

Static Checking

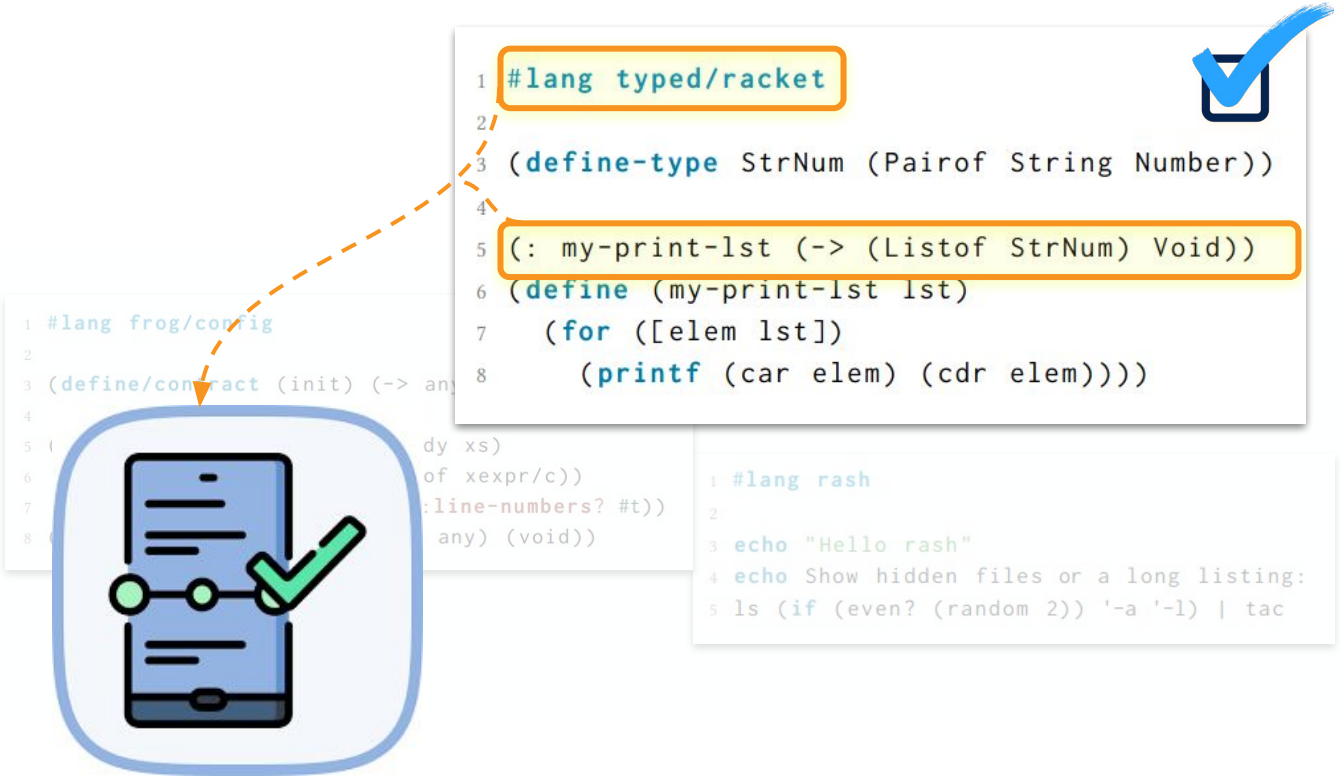


```
#lang frog/config

(define/contract (init) (-> any)
  (current-title "My Blog"))
(define/contract (enhance-body xs)
  (-> (listof xexpr/c) (listof xexpr/c))
  (~> xs (syntax-highlight #:line-numbers? #t))
(define/contract (clean) (-> any) (void))
```

```
#lang typed/racket

(define-type StrNum (Pairof String Number))

(: my-print-lst (-> (Listof StrNum) Void))
(define (my-print-lst lst)
  (for ([elem lst])
    (printf (car elem) (cdr elem))))
```

```
#lang rash

echo "Hello rash"
echo Show hidden files or a long listing:
ls (if (even? (random 2)) '-a '-l) | tac
```

Static Checking



Static Checking

```
#lang frog/config

(define/contract (init) (-> any)
  (current-title "My Blog"))
(define/contract (enhance-body xs)
  (-> (listof xexpr/c) (listof xexpr/c))
  (~> xs (syntax-highlight #:line-numbers? #t))
(define/contract (clean) (-> any) (void))
```

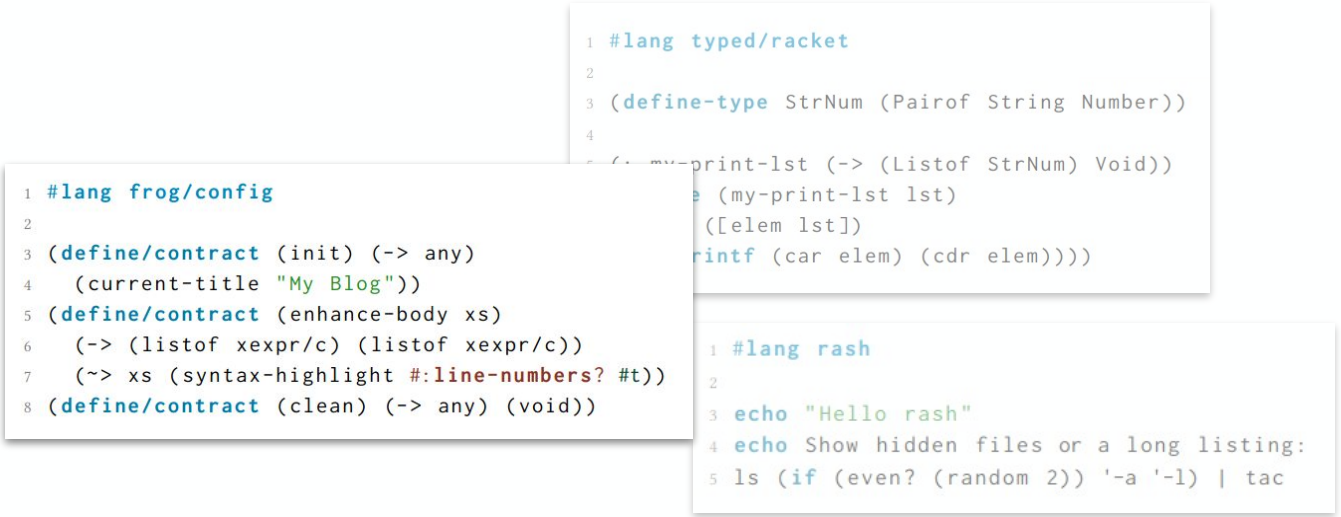
```
#lang typed/racket

(define-type StrNum (Pairof String Number))

(: my-print-lst (-> (Listof StrNum) Void))
(define (my-print-lst lst)
  (for ([elem lst])
    (printf (car elem) (cdr elem))))
```

```
#lang rash

echo "Hello rash"
echo Show hidden files or a long listing:
ls (if (even? (random 2)) '-a '-l) | tac
```

```
#lang frog/config

(define/contract (init) (-> any)
  (current-title "My Blog"))
(define/contract (enhance-body xs
  (-> (listof xexpr/c) (listof xe
  (~> xs (syntax-highlight #:line
(define/contract (clean) (-> any)
```

```
#lang typed/racket

(define-type StrNum (Pairof String Number))

(: my-print-lst (-> (Listof StrNum) Void))
(define (my-print-lst lst)
  (for ([elem lst])
    (printf (car elem) (cdr elem)))))
```

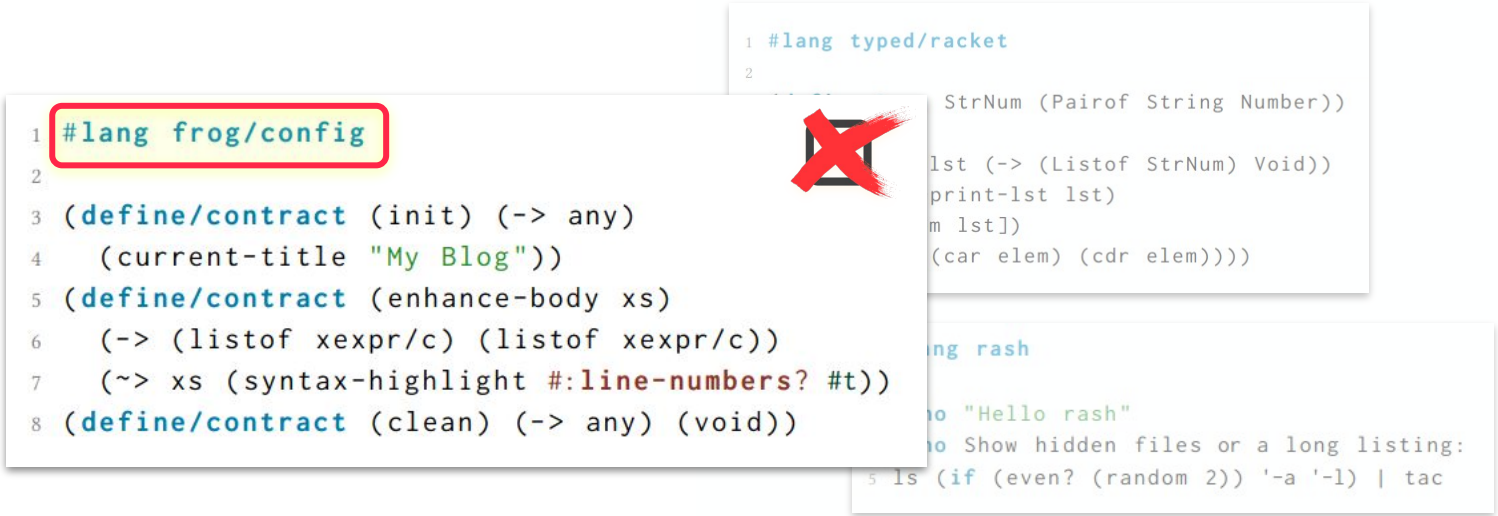
```
#lang rash

echo "Hello rash"
echo Show hidden files or a long listing:
ls (if (even? (random 2)) '-a '-l) | tac
```



```
#lang typed/racket

(define-type StrNum (Pairof String Number))

(: my-print-lst (-> (Listof StrNum) Void))
(define (my-print-lst lst)
  (for ([elem lst])
    (printf (car elem) (cdr elem))))
```

```
#lang frog/config

(define/contract (init) (-> any)
  (current-title "My Blog"))
(define/contract (enhance-body xs)
  (-> (listof xexpr/c) (listof xexpr/c))
  (~> xs (syntax-highlight #:line-numbers? #t)))
(define/contract (clean) (-> any) (void))
```

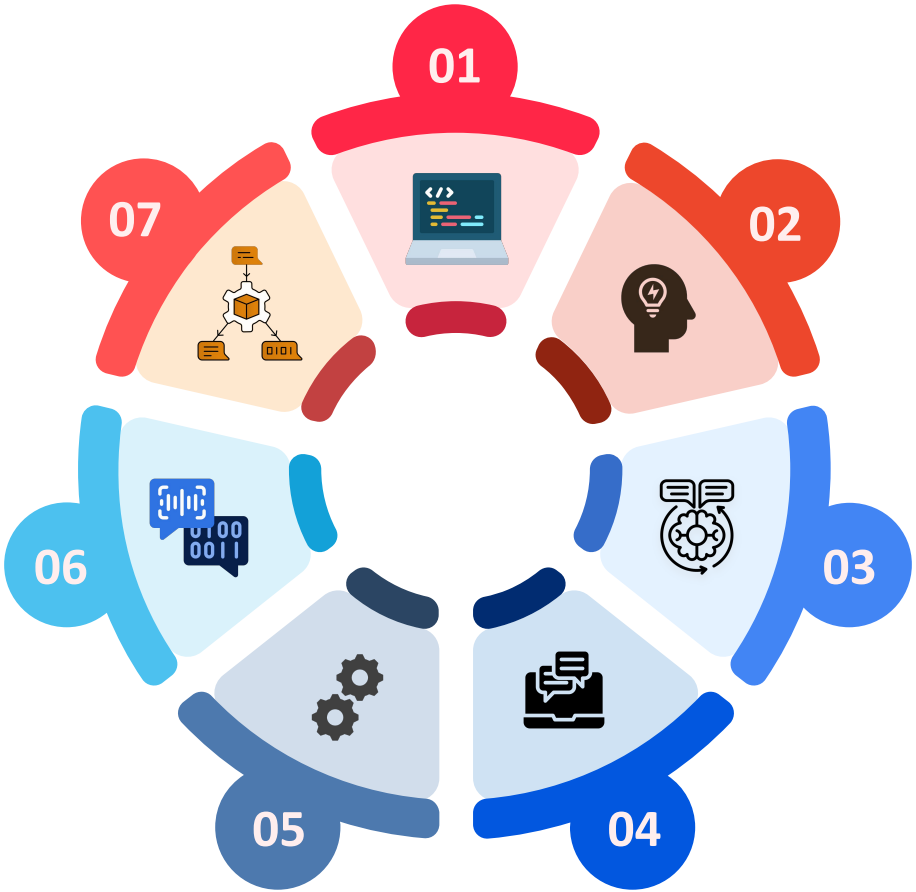
```
#lang rash

echo "Hello rash"
echo Show hidden files or a long listing:
ls (if (even? (random 2)) '-a '-l) | tac
```

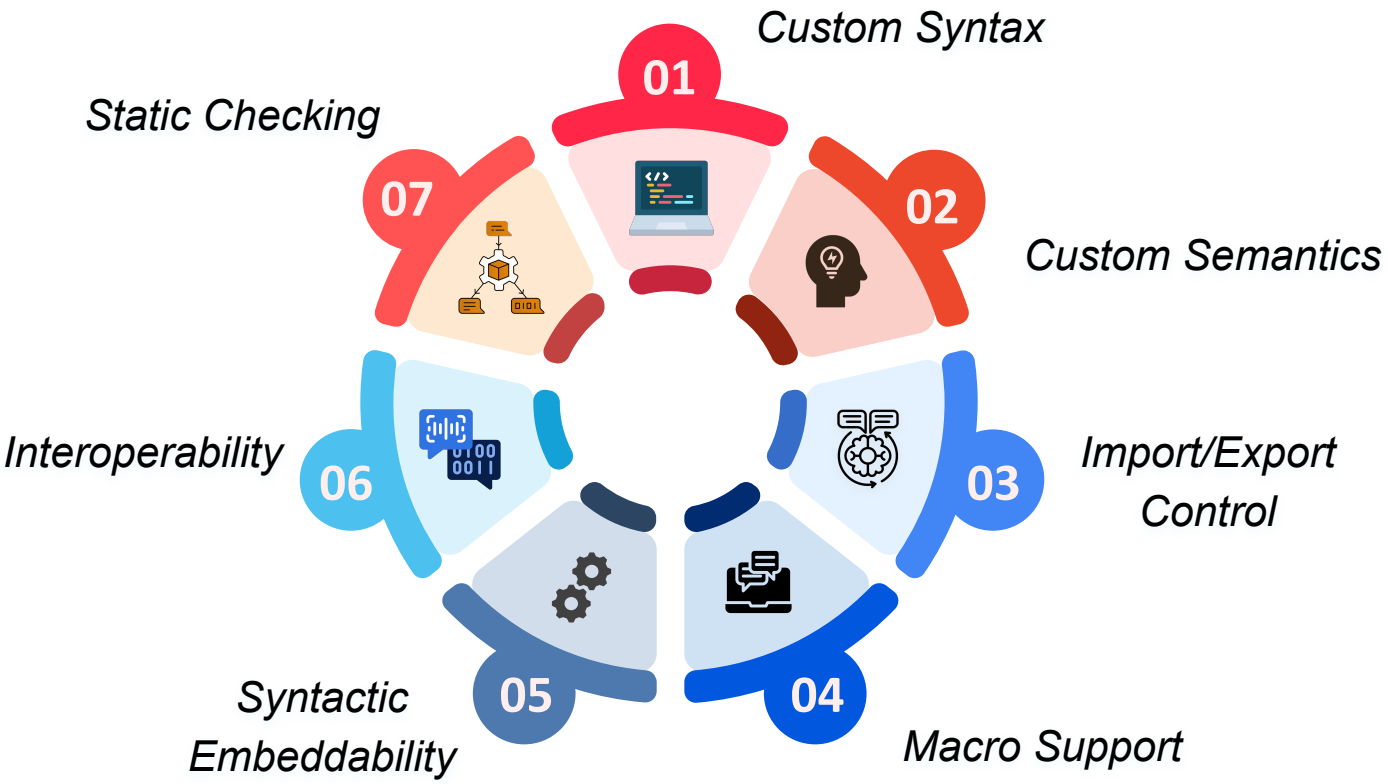

#lang frog/config
(define/contract (init) (-> any)
(current-title "My Blog"))
(define/contract (enhance-body xs)
(-> (listof xexpr/c) (listof xexpr/c))
(~> xs (syntax-highlight #:line-numbers? #t))
(define/contract (clean) (-> any) (void))
#lang typed/racket
StrNum (Pairof String Number))
lst (-> (Listof StrNum) Void))
print-lst lst)
m lst])
(car elem) (cdr elem))))
ng rash
o "Hello rash"
o Show hidden files or a long listing:
ls (if (even? (random 2)) '-a '-l) | tac

```
#lang frog/config

(define/contract (init) (-> any)
  (current-title "My Blog"))
(define/contract (enhance-body xs)
  (-> (listof xexpr/c) (listof xexpr/c))
  (~> xs (syntax-highlight #:line-numbers? #t))
(define/contract (clean) (-> any) (void))
```

```
#lang typed/racket

(define-type StrNum (Pairof String Number))

(: my-print-lst (-> (Listof StrNum) Void))
(define (my-print-lst lst)
  (for ([elem lst])
    (printf (car elem) (cdr elem))))
```

```
#lang rash

echo "Hello rash"
echo Show hidden files or a long listing:
ls (if (even? (random 2)) '-a '-l) | tac
```


01
02
03
04
05
06
07


Custom Syntax
01
Custom Semantics
02
Import/Export
Control
03
Macro Support
04
Syntactic
Embeddability
05
Interoperability
06
Static Checking
07

In the rest of the talk

- Language Design Intents
- Enabling Mechanisms
- Survey of DSLs in Racket

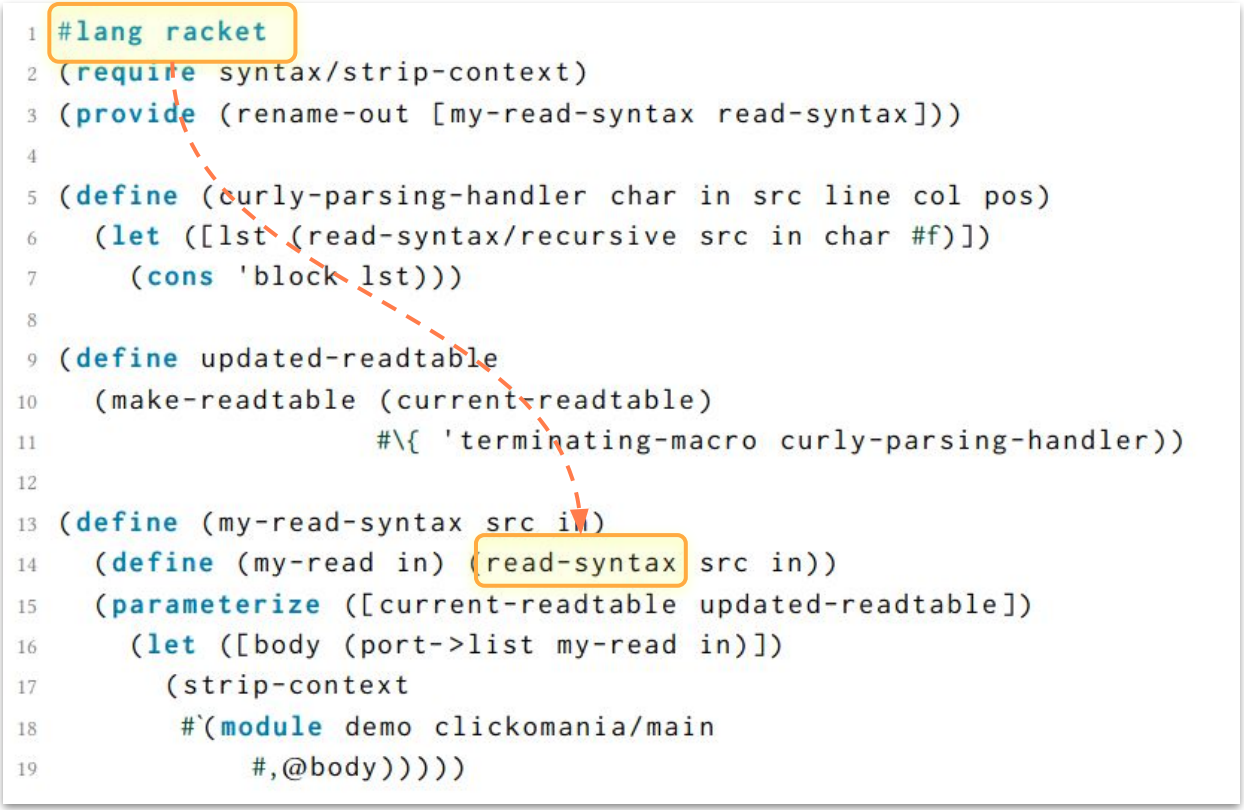
- Enabling Mechanisms

- Enabling Mechanisms
    - Reader Customization

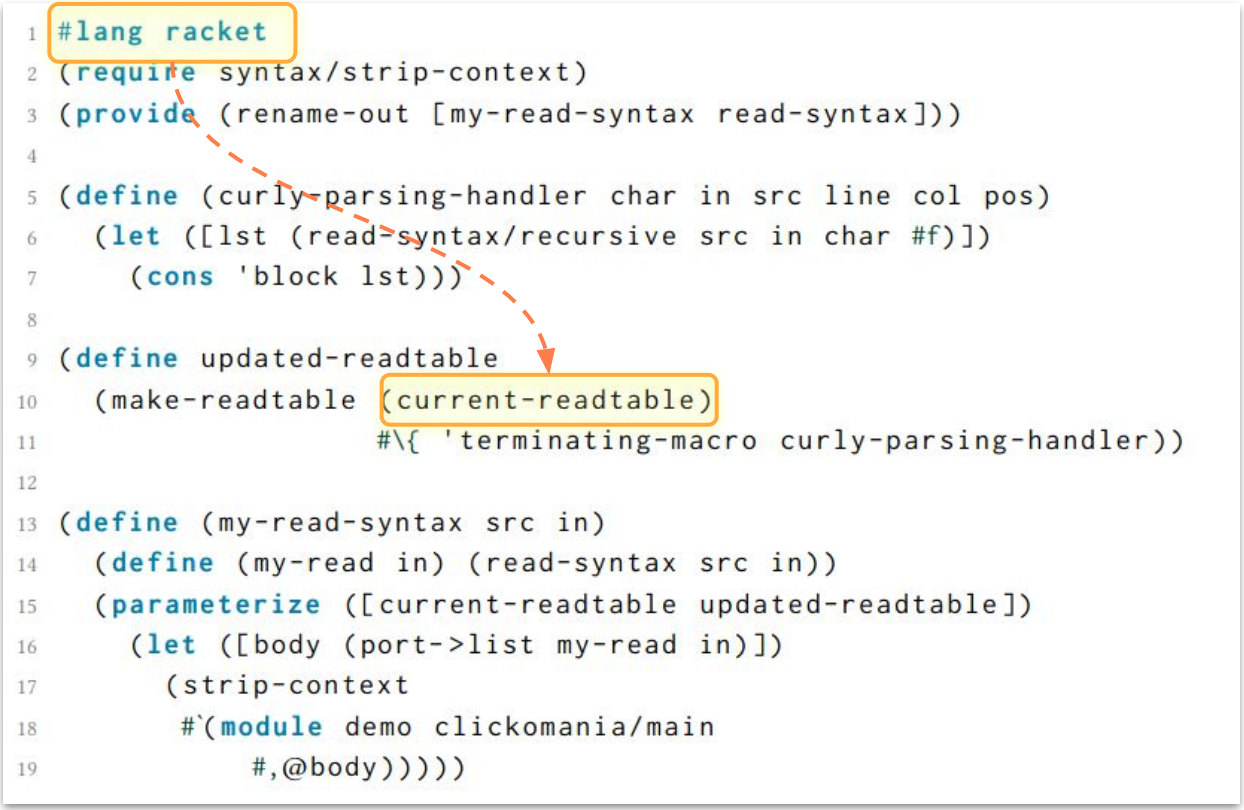
- Enabling Mechanisms
    - Reader Customization
    - Expander Customization

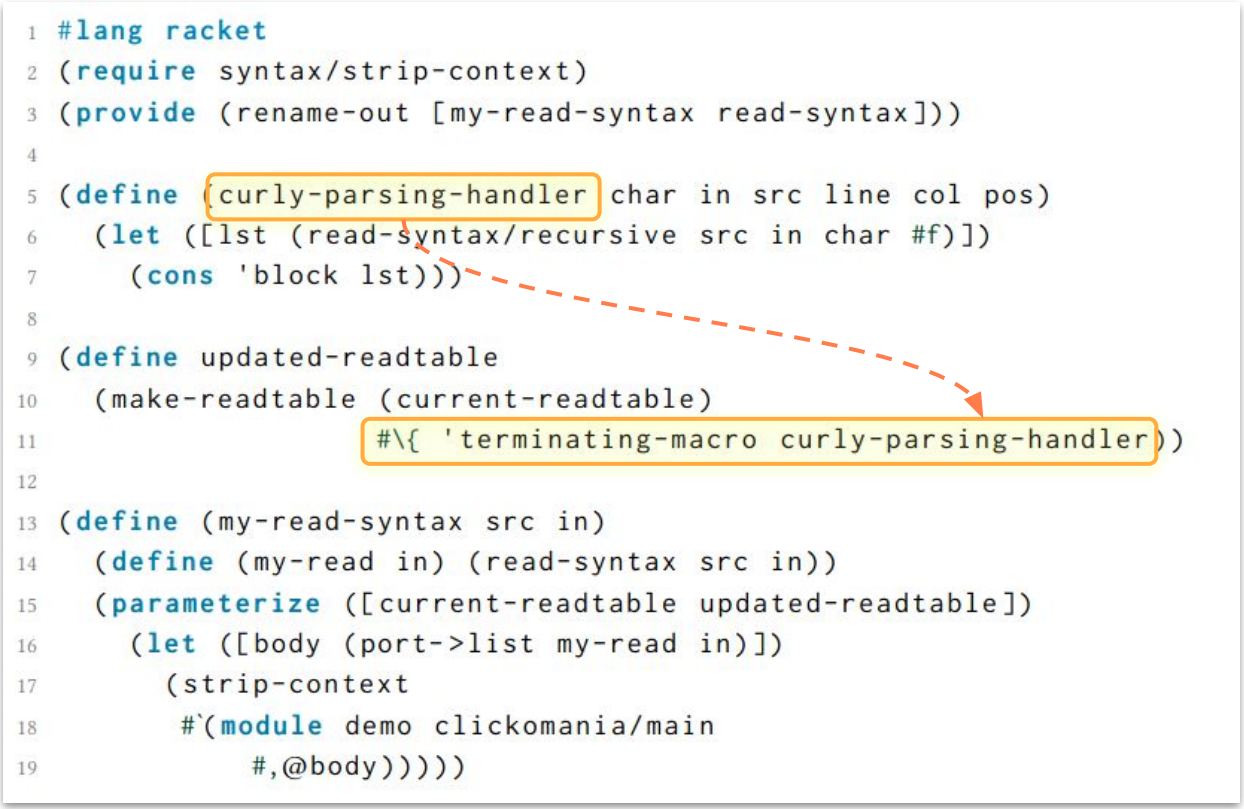
- Enabling Mechanisms
  - Reader Customization
  - Expander Customization

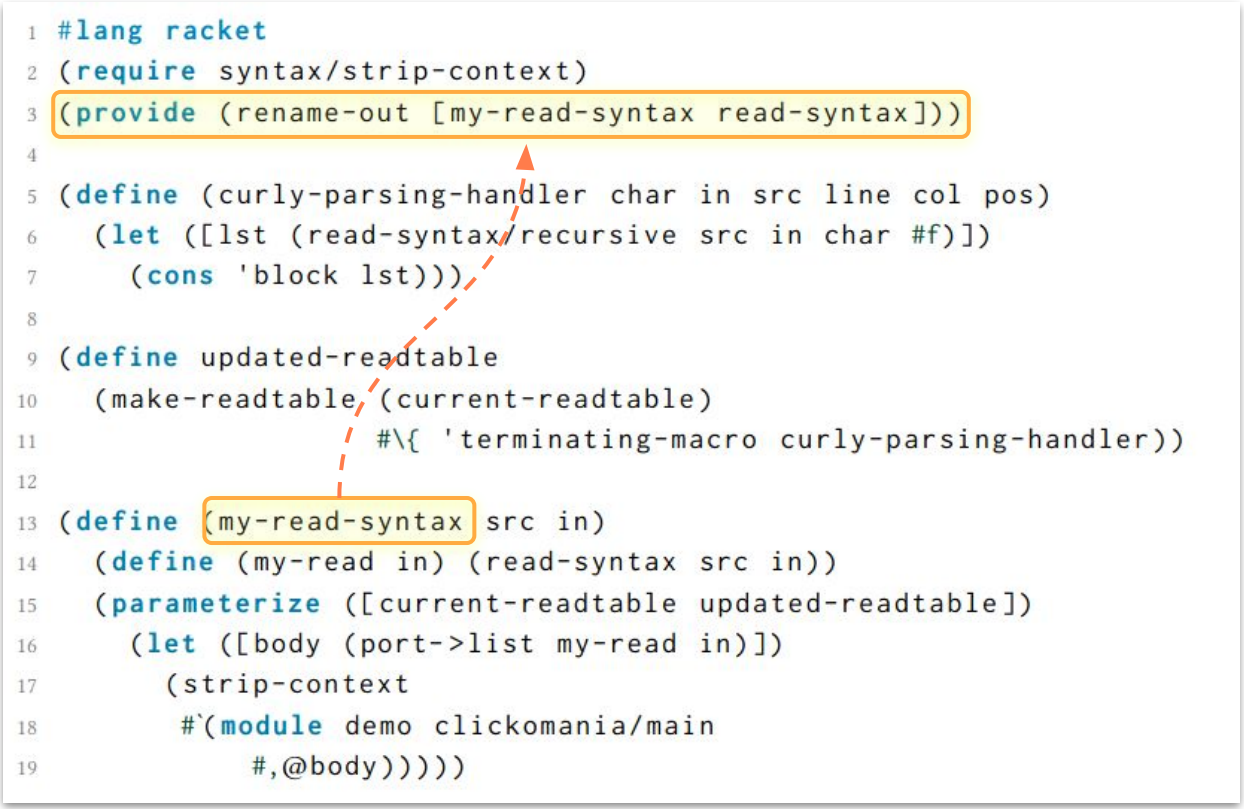
```
#lang racket
(require syntax/strip-context)
(provide (rename-out [my-read-syntax read-syntax]))

(define (curly-parsing-handler char in src line col pos)
  (let ([lst (read-syntax/recursive src in char #f)])
    (cons 'block lst)))

(define updated-readtable
  (make-readtable (current-readtable)
                  #\{ 'terminating-macro curly-parsing-handler))

(define (my-read-syntax src in)
  (define (my-read in) (read-syntax src in))
  (parameterize ([current-readtable updated-readtable])
    (let ([body (port->list my-read in)])
      (strip-context
       #`(module demo clickomania/main
           #,@body)))))
```



```
#lang racket
(require syntax/strip-context)
(provide (rename-out [my-read-syntax read-syntax]))

(define (curly-parsing-handler char in src line col pos)
  (let ([lst (read-syntax/recursive src in char #f)])
    (cons 'block lst)))

(define updated-readtable
  (make-readtable (current-readtable)
                  #\{ 'terminating-macro curly-parsing-handler))

(define (my-read-syntax src in)
  (define (my-read in) (read-syntax src in))
  (parameterize ([current-readtable updated-readtable])
    (let ([body (port->list my-read in)])
      (strip-context
       #`(module demo clickomania/main
           #,@body)))))
```


#lang racket
(require syntax/strip-context)
(provide (rename-out [my-read-syntax read-syntax]))
(define (curly-parsing-handler char in src line col pos)
(let ([lst (read-syntax/recursive src in char #f)])
(cons 'block lst)))
(define updated-readtable
(make-readtable (current-readtable)
#\{ 'terminating-macro curly-parsing-handler))
(define (my-read-syntax src in)
(define (my-read in) (read-syntax src in))
(parameterize ([current-readtable updated-readtable])
(let ([body (port->list my-read in)])
(strip-context
#`(module demo clickomania/main
#,@body)))))


#lang racket
(require syntax/strip-context)
(provide (rename-out [my-read-syntax read-syntax]))
(define (curly-parsing-handler char in src line col pos)
(let ([lst (read-syntax/recursive src in char #f)])
(cons 'block lst)))
(define updated-readtable
(make-readtable (current-readtable)
#\{ 'terminating-macro curly-parsing-handler))
(define (my-read-syntax src in)
(define (my-read in) (read-syntax src in))
(parameterize ([current-readtable updated-readtable])
(let ([body (port->list my-read in)])
(strip-context
#`(module demo clickomania/main
#,@body)))))


#lang racket
(require syntax/strip-context)
(provide (rename-out [my-read-syntax read-syntax]))
(define (curly-parsing-handler char in src line col pos)
(let ([lst (read-syntax/recursive src in char #f)])
(cons 'block lst)))
(define updated-readtable
(make-readtable (current-readtable)
#\{ 'terminating-macro curly-parsing-handler))
(define (my-read-syntax src in)
(define (my-read in) (read-syntax src in))
(parameterize ([current-readtable updated-readtable])
(let ([body (port->list my-read in)])
(strip-context
#`(module demo clickomania/main
#,@body)))))

```
#lang racket
(require syntax/strip-context)
(provide (rename-out [my-read-syntax read-syntax]))

(define (curly-parsing-handler char in src line col pos)
  (let ([lst (read-syntax/recursive src in char #f)])
    (cons 'block lst)))

(define updated-readtable
  (make-readtable (current-readtable)
                  #\{ 'terminating-macro curly-parsing-handler))

(define (my-read-syntax src in)
  (define (my-read in) (read-syntax src in))
  (parameterize ([current-readtable updated-readtable])
    (let ([body (port->list my-read in)])
      (strip-context
       #`(module demo clickomania/main
           #,@body)))))
```

```
#lang racket
(require syntax/strip-context)
(provide (rename-out [my-read-syntax read-syntax]))

(define (curly-parsing-handler char in src line col pos)
  (let ([lst (read-syntax/recursive src in char #f)])
    (cons 'block lst)))

(define updated-readtable
  (make-readtable (current-readtable)
                  #\{ 'terminating-macro curly-parsing-handler))

(define (my-read-syntax src in)
  (define (my-read in) (read-syntax src in))
  (parameterize ([current-readtable updated-readtable])
    (let ([body (port->list my-read in)])
      (strip-context
       #`(module demo clickomania/main
           #,@body)))))
```

Customized Reader

```
#lang racket
(require syntax/strip-context)
(provide (rename-out [my-read-syntax read-syntax]))

(define (curly-parsing-handler char in src line col pos)
  (let ([lst (read-syntax/recursive src in char #f)])
    (cons 'block lst)))

(define updated-readtable
  (make-readtable (current-readtable)
                  #\{ 'terminating-macro curly-parsing-handler))

(define (my-read-syntax src in)
  (define (my-read in) (read-syntax src in))
  (parameterize ([current-readtable updated-readtable])
    (let ([body (port->list my-read in)])
      (strip-context
       #`(module demo clickomania/main
           #,@body)))))
```

```
#lang clickomania

(define (create-background-image bsize bg-img-path img-path)
  (if (<= 400 bsize) {
        (define back-img (adjust-image-with-path bsize bg-img-path))
        (define img (adjust-image-with-path (* bsize 0.8) img-path))
        (define rotated-img (rotate -5 bimage))
        (underlay back-img rotated-img)
      }
      (adjust-image-with-path bsize bg-img-path)))

(define sunset-cloud (create-background-image 500 "./sunset.png" "./cloud.png"))
(set-background-image sunset-cloud)
(run 500 5 #:theme "candy" #:num-colors 4 #:num-clicks 10)
```

```
1  #lang racket
2
3  (require syntax/strip-context)
4  (provide (rename-out [my-infix-read-syntax
       read-syntax]))
5
6  (define (my-infix-read-syntax src in)
7    (let ([body (custom-parse in)])
8      (strip-context
9       #`(module any clickomania-infix/main
10          #,body))))
```

```
1  #lang racket
2
3  (require syntax/strip-context)
4  (provide (rename-out [my-infix-read-syntax
       read-syntax]))
5
6  (define (my-infix-read-syntax src in)
7    (let ([body (custom-parse in)])
8      (strip-context
9       #`(module any clickomania-infix/main
10          #,body))))
```

```
1  #lang racket
2
3  (require syntax/strip-context)
4  (provide (rename-out [my-infix-read-syntax
       read-syntax]))
5
6  (define (my-infix-read-syntax src in)
7    (let ([body (custom-parse in)])
8      (strip-context
9       #`(module any clickomania-infix/main
10          #,body))))
```

Customized Reader

```
#lang racket

(require syntax/strip-context)
(provide (rename-out [my-infix-read-syntax
    read-syntax]))

(define (my-infix-read-syntax src in)
  (let ([body (custom-parse in)])
    (strip-context
     #`(module any clickomania-infix/main
         #,body))))
```

```
#lang clickomania-infix

fun create_background_image (bsize, bg_img_path, img_path):
  if 400 <= bsize
  | def back_img = adjust_image_with_path (bsize, bg_img_path)
    def img = adjust_image_with_path (bsize * 0.8, img_path)
    underlay (back_img, img)
  | adjust_image_with_path (bsize, bg_img_path)

set_background_image (
  create_background_image (500, "./sunset.png", "./cloud.png"))
run (500, 5, ~theme: "candy", ~num_colors: 4, ~num_clicks: 10)
```

```
#lang racket

(require syntax/strip-context)
(provide (rename-out [my-infix-read-syntax
    read-syntax]))

(define (my-infix-read-syntax src in)
  (let ([body (custom-parse in)])
    (strip-context
     #`(module any clickomania-infix/main
         #,body))))
```

```
#lang clickomania-infix

fun create_background_image (bsize, bg_img_path, img_path):
  if 400 <= bsize
  | def back_img = adjust_image_with_path (bsize, bg_img_path)
    def img = adjust_image_with_path (bsize * 0.8, img_path)
    underlay (back_img, img)
  | adjust_image_with_path (bsize, bg_img_path)

set_background_image (
  create_background_image (500, "./sunset.png", "./cloud.png"))
run (500, 5, ~theme: "candy", ~num_colors: 4, ~num_clicks: 10)
```

```
#lang racket

(require syntax/strip-context)
(provide (rename-out [my-hybrid-read-syntax
                      read-syntax]))

(define (my-hybrid-read-syntax src in)
  (define (my-read x) (read-syntax src x))
  (let ([peeked (peek-char in 1)])
    (cond
      [(equal? #\( peeked)
       (let ([body (port->list my-read in)])
         (strip-context
          #`(module sexp racket
              #,@body)))]
      [else
       (let ([body (custom-parse in)])
         (strip-context
          #`(module nonsexp
    clickomania-hybrid/main
              #,body)))])))
```

```
#lang racket

(require syntax/strip-context)
(provide (rename-out [my-hybrid-read-syntax
                      read-syntax]))

(define (my-hybrid-read-syntax src in)
  (define (my-read x) (read-syntax src x))
  (let ([peeked (peek-char in 1)])
    (cond
      [(equal? #\( peeked)
       (let ([body (port->list my-read in)])
         (strip-context
          #`(module sexp racket
              #,@body)))]
      [else
       (let ([body (custom-parse in)])
         (strip-context
          #`(module nonsexp
    clickomania-hybrid/main
              #,body)))])))
```

```
#lang racket

(require syntax/strip-context)
(provide (rename-out [my-hybrid-read-syntax
                      read-syntax]))

(define (my-hybrid-read-syntax src in)
  (define (my-read x) (read-syntax src x))
  (let ([peeked (peek-char in 1)])
    (cond
      [(equal? #\( peeked)
       (let ([body (port->list my-read in)])
         (strip-context
          #`(module sexp racket
              #,@body)))]
      [else
       (let ([body (custom-parse in)])
         (strip-context
          #`(module nonsexp
    clickomania-hybrid/main
              #,body)))])))
```

Customized Reader

```
#lang racket

(require syntax/strip-context)
(provide (rename-out [my-hybrid-read-syntax
                      read-syntax]))

(define (my-hybrid-read-syntax src in)
  (define (my-read x) (read-syntax src x))
  (let ([peeked (peek-char in 1)])
    (cond
      [(equal? #\( peeked)
       (let ([body (port->list my-read in)])
         (strip-context
          #`(module sexp racket
              #,@body)))]
      [else
       (let ([body (custom-parse in)])
         (strip-context
          #`(module nonsexp
    clickomania-hybrid/main
              #,body)))])))
```

```
#lang clickomania

(define sunset (adjust-image-with-path 500 "./sunset.png"))
(set-background-image sunset)
(run 500 5 #:theme "candy" #:num-colors 4 #:num-clicks 10)
```

```
#lang racket

(require syntax/strip-context)
(provide (rename-out [my-hybrid-read-syntax
                      read-syntax]))

(define (my-hybrid-read-syntax src in)
  (define (my-read x) (read-syntax src x))
  (let ([peeked (peek-char in 1)])
    (cond
      [(equal? #\( peeked)
       (let ([body (port->list my-read in)])
         (strip-context
          #`(module sexp racket
              #,@body)))]
      [else
       (let ([body (custom-parse in)])
         (strip-context
          #`(module nonsexp
    clickomania-hybrid/main
              #,body)))])))
```

```
#lang clickomania

(define sunset (adjust-image-with-path 500 "./sunset.png"))
(set-background-image sunset)
(run 500 5 #:theme "candy" #:num-colors 4 #:num-clicks 10)
```

```
#lang clickomania

fun create_background_image (bsize, bg_img_path, img_path):
  if 400 <= bsize
  | def back_img = adjust_image_with_path (bsize, bg_img_path)
    def img = adjust_image_with_path (bsize * 0.8, img_path)
    underlay (back_img, img)
  | adjust_image_with_path (bsize, bg_img_path)

set_background_image (
  create_background_image (500, "./sunset.png", "./cloud.png"))
run (500, 5, ~theme: "candy", ~num_colors: 4, ~num_clicks: 10)
```

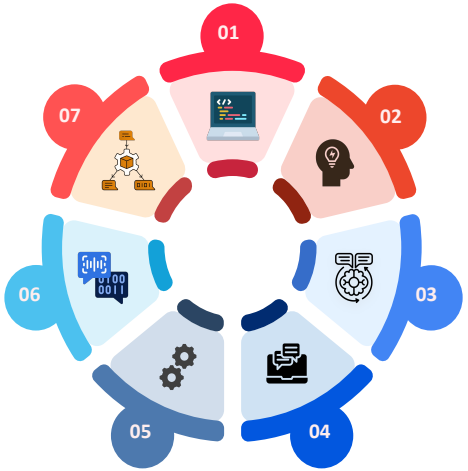
Reader Customization

Reader Customization

- *Default Reader*
- *Extended Reader*
- *Custom Reader*
- *Hybrid Reader*

Reader Customization

- *Default Reader*
- *Extended Reader*
- *Custom Reader*
- *Hybrid Reader*



Reader Customization

- *Default Reader*
- *Extended Reader*
- *Custom Reader*
- *Hybrid Reader*

Custom Syntax

- Enabling Mechanisms
  - Reader Customization
  - Expander Customization

Expander Customization

Expander Customization

- *Default Bindings*
- *Custom Bindings*
- *Default Bindings at Phase 1*
- *Custom* `#%module-begin`
- *Other Custom Implicit Forms*

Expander Environment

```
1  #lang racket
2  (require "image.rkt" "game.rkt" "sound.rkt"
3           (for-syntax racket syntax/parse))
4
5  (provide (except-out (all-from-out racket) #%module-begin)
6           (rename-out [my-module-begin #%module-begin])
7           (all-from-out "image.rkt" "game.rkt" "sound.rkt")
8           (for-syntax (all-from-out racket))
9
10 (define-syntax (my-module-begin stx)
11   (syntax-parse stx
12     [(_ form ...)
13      (with-syntax ([required-sym (format-id stx "game-info")])
14        #'(#%module-begin form ...
15                          (provide required-sym)))]))
```

```
1  #lang racket
2  (require "image.rkt" "game.rkt" "sound.rkt"
3           (for-syntax racket syntax/parse))
4
5  (provide (except-out (all-from-out racket) #%module-begin)
6           (rename-out [my-module-begin #%module-begin])
7           (all-from-out "image.rkt" "game.rkt" "sound.rkt")
8           (for-syntax (all-from-out racket))
9
10 (define-syntax (my-module-begin stx)
11   (syntax-parse stx
12     [(_ form ...)
13      (with-syntax ([required-sym (format-id stx "game-info")])
14        #'(#%module-begin form ...
15                          (provide required-sym)))]))
```

```
1  #lang racket
2  (require "image.rkt" "game.rkt" "sound.rkt"
3           (for-syntax racket syntax/parse))
4
5  (provide (except-out (all-from-out racket) #%module-begin)
6           (rename-out [my-module-begin #%module-begin])
7           (all-from-out "image.rkt" "game.rkt" "sound.rkt")
8           (for-syntax (all-from-out racket))
9
10 (define-syntax (my-module-begin stx)
11   (syntax-parse stx
12     [(_ form ...)
13      (with-syntax ([required-sym (format-id stx "game-info")])
14        #'(#%module-begin form ...
15                          (provide required-sym)))]))
```

```
1  #lang racket
2  (require "image.rkt" "game.rkt" "sound.rkt"
3           (for-syntax racket syntax/parse))
4
5  (provide (except-out (all-from-out racket) #%module-begin)
6           (rename-out [my-module-begin #%module-begin])
7           (all-from-out "image.rkt" "game.rkt" "sound.rkt")
8           (for-syntax (all-from-out racket)))
9
10 (define-syntax (my-module-begin stx)
11   (syntax-parse stx
12     [(_ form ...)
13      (with-syntax ([required-sym (format-id stx "game-info")])
14        #'(#%module-begin form ...
15                          (provide required-sym)))]))
```

Custom `#%module-begin`

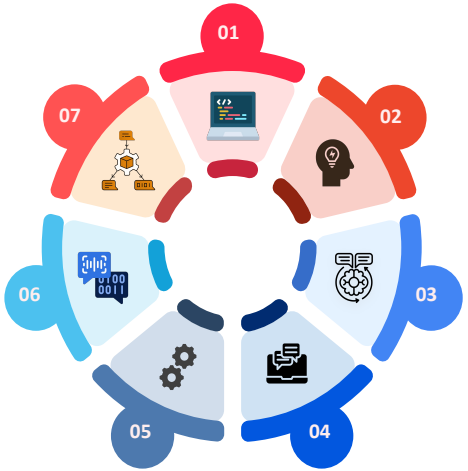
```
1  #lang racket
2  (require "image.rkt" "game.rkt" "sound.rkt"
3           (for-syntax racket syntax/parse))
4
5  (provide (except-out (all-from-out racket) #%module-begin)
6           (rename-out [my-module-begin #%module-begin])
7           (all-from-out "image.rkt" "game.rkt" "sound.rkt")
8           (for-syntax (all-from-out racket))
9
10 (define-syntax (my-module-begin stx)
11   (syntax-parse stx
12     [(_ form ...)
13      (with-syntax ([required-sym (format-id stx "game-info")])
14        #'(#%module-begin form ...
15                          (provide required-sym)))]))
```

- *Default Reader*
- *Extended Reader*
- *Custom Reader*
- *Hybrid Reader*

- *Default Bindings*
- *Custom Bindings*
- *Default Bindings at Phase 1*
- *Custom* `#%module-begin`
- *Other Custom Implicit Forms*

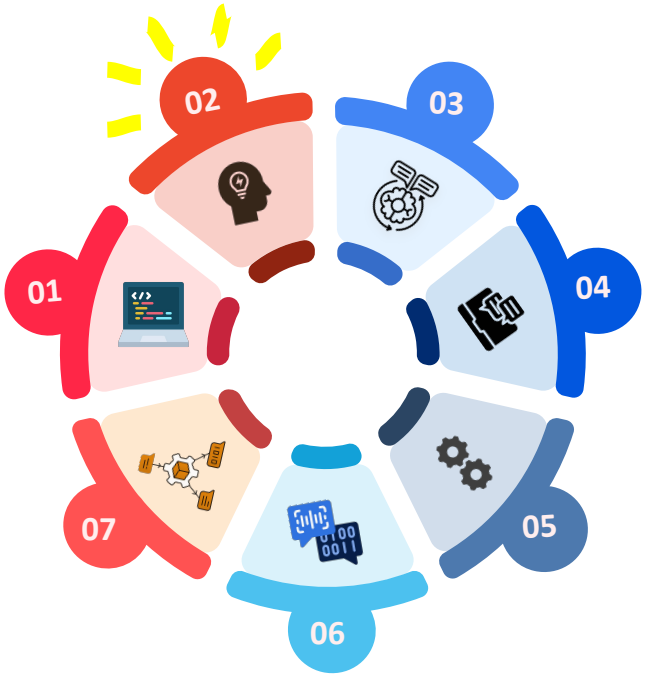
- *Default Reader*
- *Extended Reader*
- *Custom Reader*
- *Hybrid Reader*

- *Default Bindings*
- *Custom Bindings*
- *Default Bindings at Phase 1*
- *Custom* `#%module-begin`
- *Other Custom Implicit Forms*

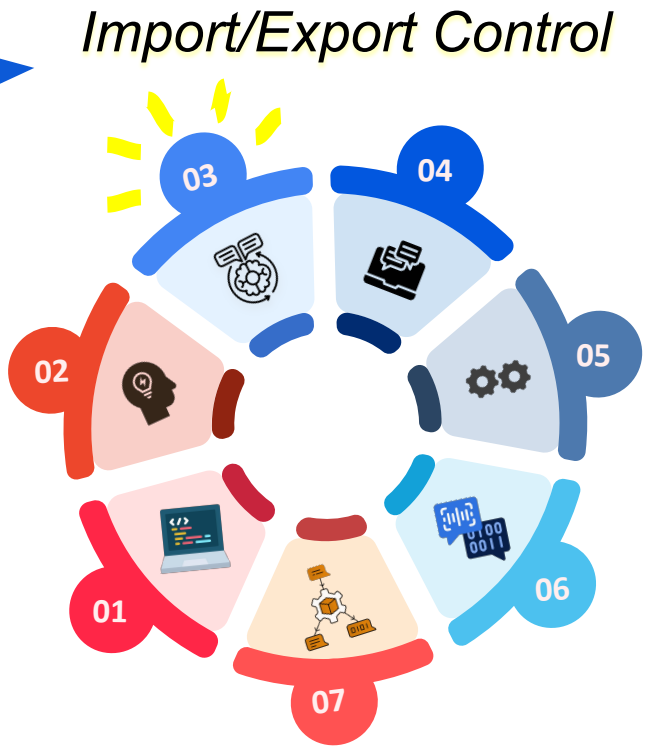


- *Default Reader*
- *Extended Reader*
- *Custom Reader*
- *Hybrid Reader*
- *Default Bindings*
- *Custom Bindings*
- *Default Bindings at Phase 1*
- *Custom #%module-begin*
- *Other Custom Implicit Forms*

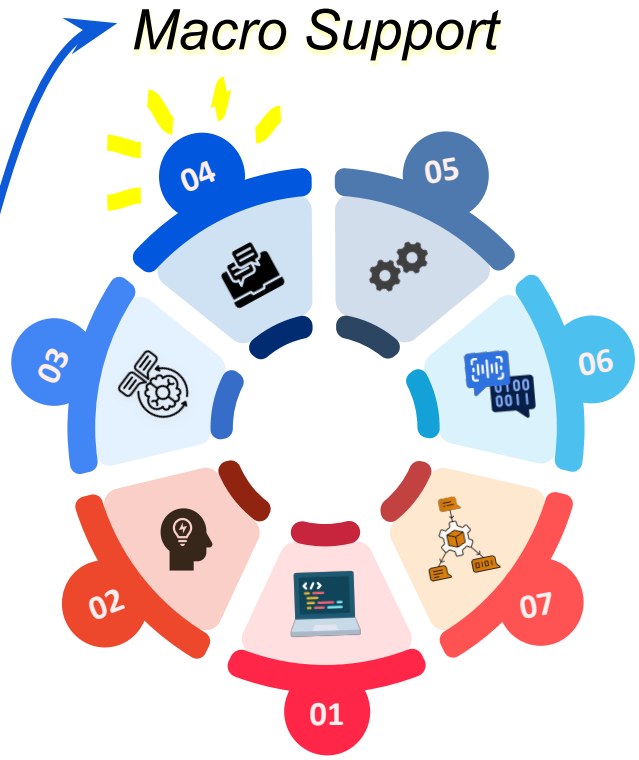
Custom Semantics



- *Default Reader*
- *Extended Reader*
- *Custom Reader*
- *Hybrid Reader*
- *Default Bindings*
- *Custom Bindings*
- *Default Bindings at Phase 1*
- *Custom* `#%module-begin`
- *Other Custom Implicit Forms*



- *Default Reader*
- *Extended Reader*
- *Custom Reader*
- *Hybrid Reader*
- *Default Bindings*
- *Custom Bindings*
- *Default Bindings at Phase 1*
- *Custom `#%module-begin`*
- *Other Custom Implicit Forms*

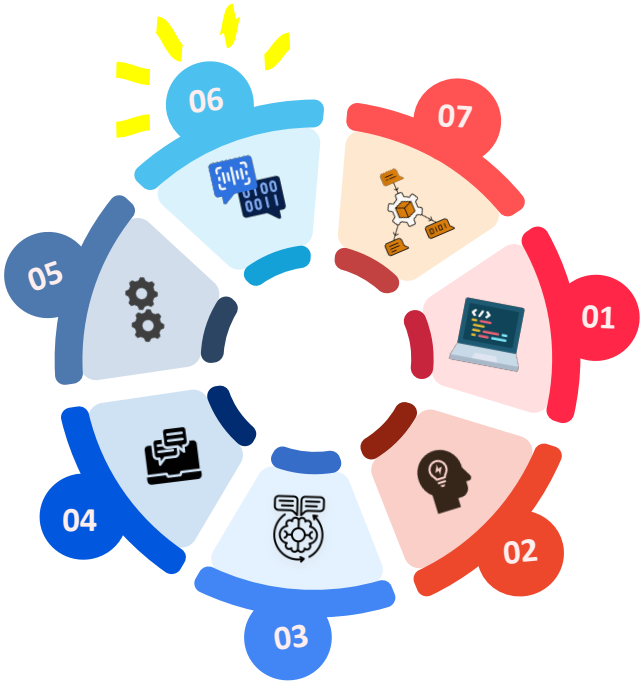


- *Default Reader*
- *Extended Reader*
- *Custom Reader*
- *Hybrid Reader*
- *Default Bindings*
- *Custom Bindings*
- *Default Bindings at Phase 1*
- *Custom `#%module-begin`*
- *Other Custom Implicit Forms*

Syntactic Embeddability

- *Default Reader*
- *Extended Reader*
- *Custom Reader*
- *Hybrid Reader*
- *Default Bindings*
- *Custom Bindings*
- *Default Bindings at Phase 1*
- *Custom* `#%module-begin`
- *Other Custom Implicit Forms*

Interoperability



- *Default Reader*
- *Extended Reader*
- *Custom Reader*
- *Hybrid Reader*
- *Default Bindings*
- *Custom Bindings*
- *Default Bindings at Phase 1*
- *Custom* `#%module-begin`
- *Other Custom Implicit Forms*

In the rest of the talk

- Language Design Intents
- Enabling Mechanisms
- Survey of DSLs in Racket

DSL	Specification	Stars	DSL	Specification	Stars
anarki	(require anarki)	1149	malt	(require malt)	137
hackett	#lang hackett	1137	turnstile	#lang turnstile	129
frog/config	#lang frog/config	906	video	#lang video	123
pie	#lang pie	661	racket-clojure	#lang clojure	118
rosette	#lang rosette	618	sketching	#lang sketching	106
rash	#lang rash	529	racket-r7rs	#lang r7rs	92
typed-racket	#lang typed/racket	494	redex	(require redex)	89
urlang	(require urlang)	296	minipascal	#lang minipascal	88
rhombus	#lang rhombus	286	algebraic	#lang algebraic	73
beautiful-racket	#lang br	282	brag	#lang brag	62
rackjure	#lang rackjure	234	sham	(require sham)	66
cur	#lang cur	215	lens	(require lens)	73
scribble	#lang scribble	189	heresy	#lang heresy	69
nanopass	#lang nanopass	174	qi	(require qi)	51
cKanren	#lang cKanren	152	racket-lua	#lang lua	50

Own Custom Syntax < Extended Racket Syntax

Own Custom Syntax < Extended Racket Syntax

	Default	Extended	Custom	Hybrid
Number of DSLs	19	6	4	1
Percentage of DSLs	63.3%	20.0%	13.3%	3.3%

Own Custom Syntax < Extended Racket Syntax

	Default	Extended	Custom	Hybrid
Number of DSLs	19	6	4	1
Percentage of DSLs	63.3%	20.0%	13.3%	3.3%

No Change of Implicit Forms

< *Change of Implicit Forms*

No Change of Implicit Forms

< Change of Implicit Forms

	Default bindings	Subset of default bindings	Custom bindings	Default bindings at phase 1	Custom `#%module-begin`	Other custom implicit forms
Number of DSLs	18	6	30	5	11	13
Percentage of DSLs	60.0%	20.0%	100.0%	16.7%	36.7%	43.3%

No Change of Implicit Forms

< Change of Implicit Forms

	Default bindings	Subset of default bindings	Custom bindings	Default bindings at phase 1	Custom `#%module-begin`	Other custom implicit forms
Number of DSLs	18	6	30	5	11	13
Percentage of DSLs	60.0%	20.0%	100.0%	16.7%	36.7%	43.3%

Without Macro Support < *With Macro Support*

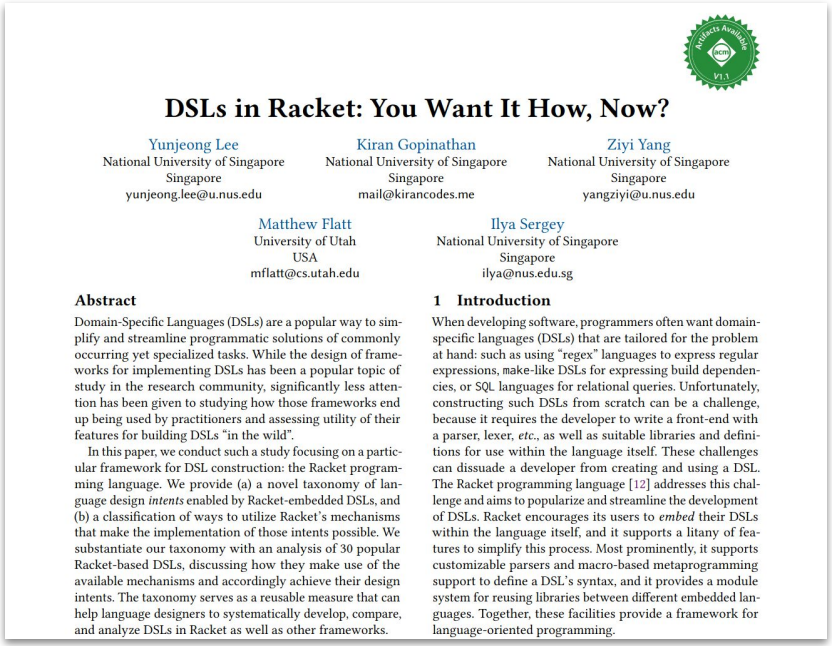
Without Macro Support < With Macro Support

	Custom Syntax	Custom Semantics	Import/Export Control	Macro Support	Embed-dability	Interoper-ability	Static Checking
Number of DSLs	11	30	12	20	21	25	6
Percentage of DSLs	36.7%	100.0%	40.0%	66.7%	70.0%	83.3%	20.0%

Without Macro Support < *With Macro Support*

	Custom Syntax	Custom Semantics	Import/Export Control	Macro Support	Embed-dability	Interoper-ability	Static Checking
Number of DSLs	11	30	12	20	21	25	6
Percentage of DSLs	36.7%	100.0%	40.0%	66.7%	70.0%	83.3%	20.0%

To Take Away



DSLs in Racket: You Want It How, Now?

Yunjeong Lee
National University of Singapore
Singapore
yunjeong.lee@u.nus.edu

Kiran Gopinathan
National University of Singapore
Singapore
mail@kirancodes.me

Ziyi Yang
National University of Singapore
Singapore
yangziyi@u.nus.edu

Matthew Flatt
University of Utah
USA
mflatt@cs.utah.edu

Ilya Sergey
National University of Singapore
Singapore
ilya@nus.edu.sg

Abstract

Domain-Specific Languages (DSLs) are a popular way to simplify and streamline programmatic solutions of commonly occurring yet specialized tasks. While the design of frameworks for implementing DSLs has been a popular topic of study in the research community, significantly less attention has been given to studying how those frameworks end up being used by practitioners and assessing utility of their features for building DSLs "in the wild".

In this paper, we conduct such a study focusing on a particular framework for DSL construction: the Racket programming language. We provide (a) a novel taxonomy of language design *intents* enabled by Racket-embedded DSLs, and (b) a classification of ways to utilize Racket's mechanisms that make the implementation of those intents possible. We substantiate our taxonomy with an analysis of 30 popular Racket-based DSLs, discussing how they make use of the available mechanisms and accordingly achieve their design intents. The taxonomy serves as a reusable measure that can help language designers to systematically develop, compare, and analyze DSLs in Racket as well as other frameworks.

1 Introduction

When developing software, programmers often want domain-specific languages (DSLs) that are tailored for the problem at hand: such as using "regex" languages to express regular expressions, `make`-like DSLs for expressing build dependencies, or `SQL` languages for relational queries. Unfortunately, constructing such DSLs from scratch can be a challenge, because it requires the developer to write a front-end with a parser, lexer, *etc.*, as well as suitable libraries and definitions for use within the language itself. These challenges can dissuade a developer from creating and using a DSL. The Racket programming language [12] addresses this challenge and aims to popularize and streamline the development of DSLs. Racket encourages its users to *embed* their DSLs within the language itself, and it supports a litany of features to simplify this process. Most prominently, it supports customizable parsers and macro-based metaprogramming support to define a DSL's syntax, and it provides a module system for reusing libraries between different embedded languages. Together, these facilities provide a framework for language-oriented programming.

- We proposed a *taxonomy of language design intents* as well as their *enabling mechanisms* in Racket's ecosystem.
- We analyzed *Racket-embedded DSLs* and made observations about how *Racket's metaprogramming facilities* are used to achieve design intents.

Thank you!