Racket：程式語言的程式語言

游書泓　PLT @ Northwestern University

Northwestern

這份投影片的想法參考自：

- Matthias Felleisen et al., *A Programmable Programming Language*, CACM
- Robby Findler, *Racket: A Programming-Language Programming Language*, Lambda Jam 2015
- Sam Tobin-Hochstadt (Originally by Robby Findler), *A picture showing all the languages used to implement Racket*
- 更多： Racket Summer School 2018 ， 可以看 “ detailed schedule here ”.

程式語言，到處都是程式語言 - 領域專門語言 (DSL)

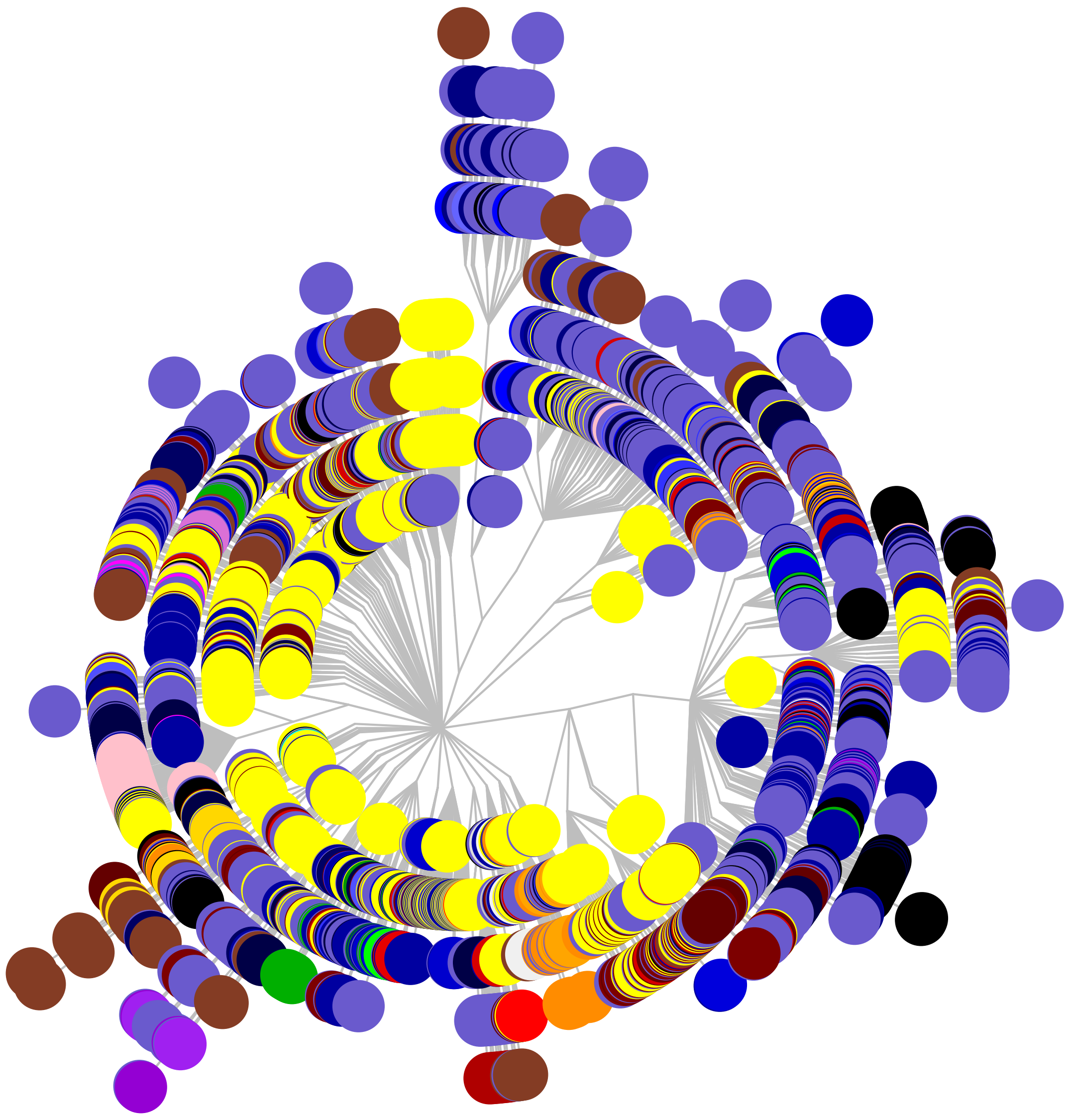
```
#lang racket

(require (for-syntax racket/base syntax/parse racket/contract))
(provide :>     (for-syntax reverse-order))

(begin-for-syntax
  (define/contract (reverse-order stx) (-> syntax? syntax?)
    (syntax-parse stx
      [() (syntax values)]
      [(f1:expr f2:expr ...)
       (with-syntax ([revd (reverse-order (syntax (f2 ...)))])
         (syntax (λ (v) (revd (f1 v)))))])))

(define-syntax (:> stx)
  (syntax-parse stx
    [(_ init-value:expr es:expr ...)
     (with-syntax ([revd (reverse-order (syntax (es ...)))])
       (syntax (let ([v init-value]
                     [f revd])
                 (f v))))]))
```

程式語言，到處都是程式語言 - 多語言並存的環境



- ● mzscheme
- ● racket/unit
- ● #%kernel
- ● racket
- ● pre-base.rkt
- ● racket/gui
- ● racket/load
- ● racket/kernel
- ● racket/private/provider
- ● racket/private/base
- ● racket/signature
- ● slideshow
- ● racket/mzscheme
- ● racket/base
- ● scribble/manual
- ● scribble/doc
- ● scribble/base
- ● scribble/base/reader
- ● at-exp racket
- ● at-exp racket/base
- ● at-exp scheme/base
- ● scribble/lp
- ● typed/racket/no-check
- ● typed/racket/base
- ● typed/scheme/base
- ● typed/racket
- ● typed-scheme/no-check
- ● typed-racket/minimal
- ● htdp/isl+
- ● htdp/bsl
- ● htdp/asl
- ● deinprogramm/DMdA
- ● htdp/isl
- ● frtime/frtime-lang-only
- ● frtime/lang-utils
- ● frtime
- ● r6rs
- ● r5rs
- ● srfi/provider
- ● swindle/turbo
- ● swindle/base
- ● web-server/insta
- ● web-server/base
- ● web-server
- ● meta/web
- ● setup/infotab
- ● syntax/module-reader
- ● everything else

Contract 的 DSL

監督服務提供者以及使用者之間的協議被正確執行

```
> (module ctc racket
   (provide
    (contract-out
      [findf (-> int-predicate/c (listof integer?)
                 int-option/c)]))

    (define int-predicate/c (-> integer? boolean?))

    (define int-option/c    (or/c #f integer?))

    (define ne-intlist/c    (and/c (listof integer?)
                                   (not/c null?))))
```

模組的 provide 及 require DSL

設定一個模組 require 的函式庫和 provide 的 identifier

```
> (module m racket
    (provide (contract-out [even? (-> any/c boolean?)])
             (struct-out pt)
             (rename-out [pt3 3d-pt]
                         [pt3? 3d-pt?])
             (except-out (all-from-out racket/control)
                         call/prompt abort/cc))

    (require racket/control)
    (struct pt (x y))
    (struct pt3 (x y z)))

  (require (rename-in (except-in (prefix-in m: 'm)
                                 m:shift0 m:reset0)
                      [m:shift shift]
                      [m:reset reset]))
```

Syntax Pattern 的 DSL

背後的核心語言特性：macro

```
> (define current-indent (make-parameter "| "))
  (define-syntax (define/trace stx)
    (syntax-parse stx
      [(_ (f arg:id ...) body:expr ...)
       (define fmt-args (apply string-append
                          (stx-map (λ (_) " ~a") (syntax (arg ...)))))
       (define fmt-pre (string-append "~a (~a" fmt-args ")\n"))
       (define fmt-post (string-append "~a (~a" fmt-args ") = ~a\n"))
       (quasisyntax
        (define (f arg ...)
          (parameterize ([current-indent
                          (string-append (current-indent) "....")])
            (printf '(unsyntax fmt-pre) (current-indent) 'f arg ...)
            (define result (let () body ...))
            (printf '(unsyntax fmt-post) (current-indent)
                    'f arg ... result)
            result)))]))

  (define/trace (fact n)
    (cond [(zero? n) 1]
          [else (* n (fact (- n 1)))]))
```

多種程式語言並存

並不僅僅是嵌入式 DSL： `racket/base` V.S. `'#%kernel` 以 keyword arguments 為例

```
> (define f
    (λ (n #:check-exact? [check? #f])
      (when check?
        (unless (exact? n)
          (printf "f: expected an exact number, but got ~a\n" n)))
      (* n 2)))

  (f 7.0)
  (f 2.3 #:check-exact? #t)
  (#%app f 2.3 #:check-exact? #t)

  (module small '#%kernel
    (define-values (f) (lambda (n val) 5))
    (#%app f 3 5))
```

撰寫程式語言的程式語言

在 module 最右上角的 **`racket/base`** 指定 *module language* ，帶入最初始的 bindings

racket/base (roughly)

```
(module base "pre-base.rkt"

  (#%require "hash.rkt"
             "list.rkt" ; shadows `reverse', `mem{q,v,ber}'
             "string.rkt"
             "stxcase-scheme.rkt"
             "qqstx.rkt"
             "stx.rkt"
             "kw-file.rkt"
             "namespace.rkt"
             "struct.rkt"
             "cert.rkt"
             "submodule.rkt"
             "generic-interfaces.rkt"
             "kw-syntax-local.rkt" ; shadows local-expand and variants
             (for-syntax "stxcase-scheme.rkt"))

  (#%provide (all-from-except "pre-base.rkt"
                              open-input-file
                              open-output-file
                              open-input-output-file
                              call-with-input-file
                              call-with-output-file
                              with-input-from-file
                              with-output-to-file
                              directory-list
                              regexp-replace*
                              new-apply-proc)
             struct
             (all-from-except "hash.rkt" paired-fold)
             (all-from "list.rkt")
             (all-from-except "string.rkt"
                              -regexp-replace*)
             (rename -regexp-replace* regexp-replace*)
             identifier?
             (all-from-except "stxcase-scheme.rkt" datum datum-case with-datum)
             (all-from-except "qqstx.rkt" quasidatum undatum undatum-splicing)
             (all-from "namespace.rkt")
             (all-from "cert.rkt")
             (all-from "submodule.rkt")
             (all-from "generic-interfaces.rkt")
             (all-from "kw-syntax-local.rkt")
             (for-syntax syntax-rules syntax-id-rules ... _)
             (rename -open-input-file open-input-file)
             (rename -open-output-file open-output-file)
             (rename -open-input-output-file open-input-output-file)
             (rename -call-with-input-file call-with-input-file)
             (rename -call-with-output-file call-with-output-file)
             (rename -with-input-from-file with-input-from-file)
             (rename -with-output-to-file with-output-to-file)
             (rename -directory-list directory-list)
             call-with-input-file*
             call-with-output-file*))
```

⬇

modlangdemo.rkt

```
#lang racket/base

(require racket/list)

(+ 1 2)
```

➡

```
(module modlangdemo racket/base
 (require racket/list)
 (+ 1 2))
```

Racket 在核心的語言中留下了「介入點」

#%module-begin #%top-interaction #%app #%top #%datum ...

```
(module modlangdemo racket/base
  (require "silent-unbound.rkt")
  (+ 1 2)
  (displayln x))
```

➡

```
(module modlangdemo racket/base
  (#%module-begin
   (module configure-runtime '#%kernel
     (#%module-begin
      (#%require racket/runtime-config)
      (#%app configure '#f)))
   (#%require "silent-unbound.rkt")
   (#%app
    call-with-values
    (lambda () (#%app + '1 '2))
    print-values)
   (#%app
    call-with-values
    (lambda () (#%app displayln 'x))
    print-values)))
```

撰寫文件用的程式語言

```
#lang scribble/base

@title{Towards a Theory of Lorem Ipsum Structure}

This is an example document. For more information,
please visit @hyperlink["https://127.0.0.1"]{our website}.

@section[#:tag "sec:abstr"]{Abstract}

@itemlist[
  @item{Lorem ipsum dolor sit amet,
        consectetur adipiscing elit,}

  @item{sed do eiusmod tempor incididunt ut
        labore et dolore magna aliqua.}
]
```

歷史上重要的語言！

```
#lang algol60

begin
    comment
      compute √x / √y assuming one of x and y is negative;
    procedure iSqrtDivSqrt(x,y);
      integer x, y;
    begin
      if x < 0 & y > 0 then
        iSqrtDivSqrt := sqrt(-1)*sqrt(-x/y)
      else if x > 0 & y < 0 then
        iSqrtDivSqrt := -sqrt(-1)*sqrt(-x/y)
    end;
    println(iSqrtDivSqrt(8,-2))
  end
```

有型別的語言

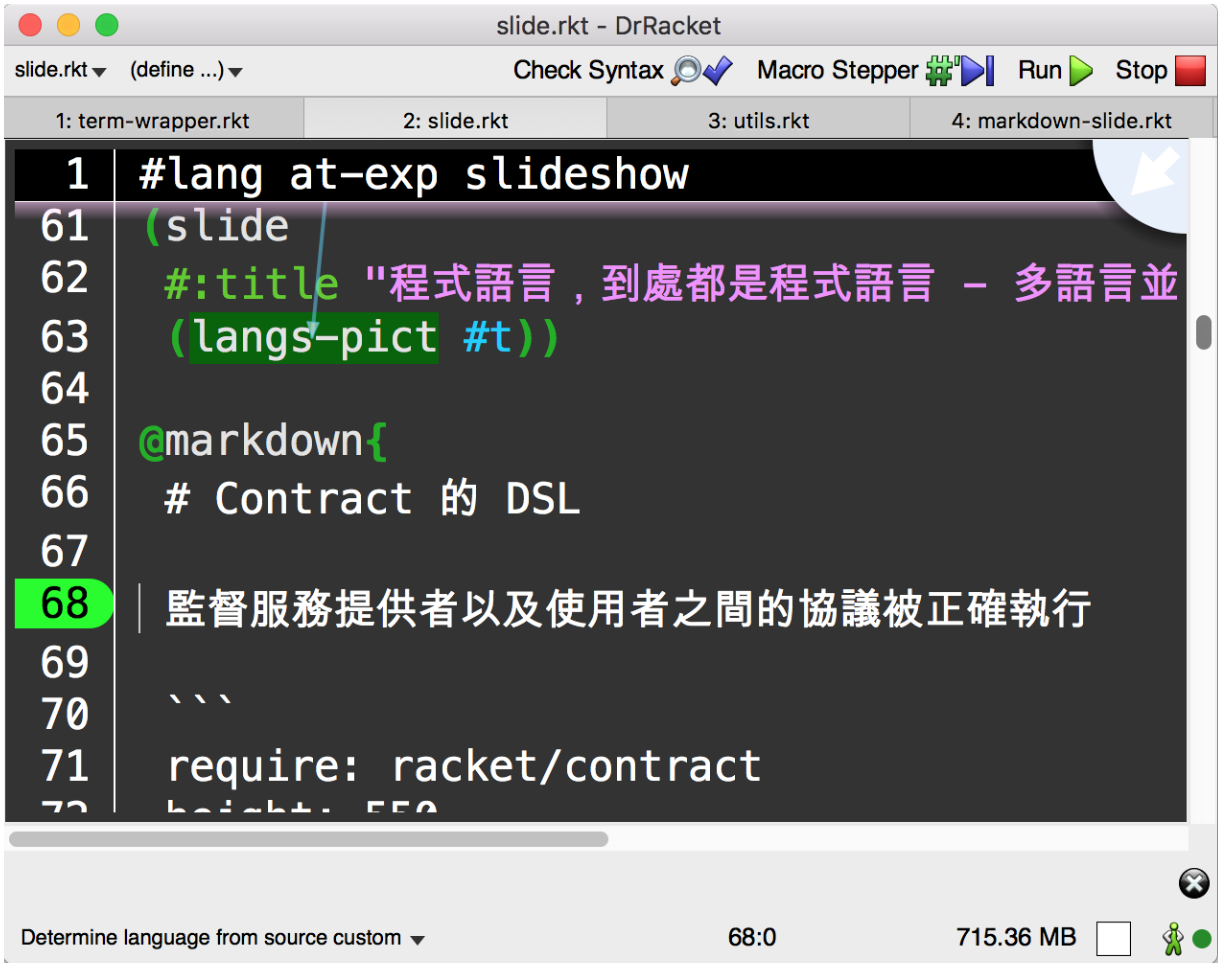
```
#lang typed

(: fact (Integer . -> . Integer))
(define (fact n)
  (cond [(zero? n) 1]
        [else (* n (fact (- n 1)))]))

(: idx-of (String (Listof String) . -> . (U #f Integer)))
(define (idx-of needle haystack)
  (cond [(null? haystack) #f]
        [(equal? needle (car haystack)) 0]
        [(idx-of needle (cdr haystack)) => (λ (idx) (+ 1 idx))]
        [else #f]))
```

語言間的互動：共享資訊

不同的語言可以在展開到 `'#%kernel`
語言的過程中留下資訊，讓使用者能實作跨語言的工具。
Check Syntax 在 DrRacket 裡畫的 binding arrows：



```
#lang at-exp slideshow
(slide
 #:title "程式語言，到處都是程式語言 – 多語言並
 (langs-pict #t))

@markdown{
 # Contract 的 DSL

 監督服務提供者以及使用者之間的協議被正確執行

 ```
 require: racket/contract
```
```

語言間的互動：保持不變量

vec.rkt

```
#lang typed/racket

(provide v-or)

(: v-or ((Vectorof Integer)
         (Integer . -> . Boolean)
         . -> .
         Boolean))
(define (v-or arr pred)
  (for/or ([n (in-vector arr)])
    (pred (vector-ref arr n))))
```

⟺

```
(-> (vectorof intger?)
    (-> integer? boolean?)
   boolean?)
```

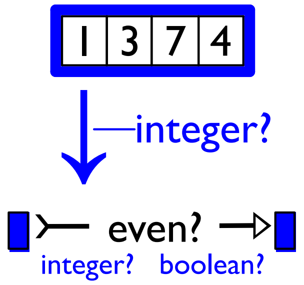
untypd.rkt

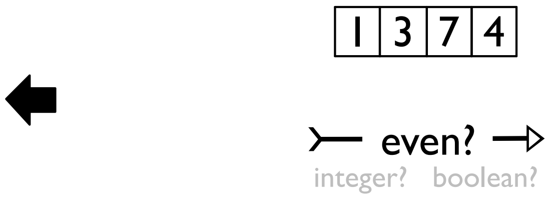
```
#lang racket/base

(provide arr)
(require "vec.rkt")

(define arr
  (vector 1 3 7 4))

(v-or arr even?)
```





API 設計拓展語言的功能

```
#lang racket/base

(provide install-loggers)
(require racket/pretty racket/format)

(define (log-compile real-compile)
  (λ (stx imm-eval?)
    (parameterize ([pretty-print-columns 50])
      (printf "=> compiling ")
      (pretty-print (syntax->datum stx)))
    (real-compile stx imm-eval?)))

(define (log-load/uc real-load/uc)
  (λ (path names)
    (printf "[loading ~a]\n"
            (~a #:max-width 50 #:limit-prefix? #t #:limit-marker "..."
                (path->string path)))
    (real-load/uc path names)))

(define (install-loggers)
  (current-load/use-compiled (log-load/uc (current-load/use-compiled)))
  (current-compile (log-compile (current-compile))))
```

更多的 API

- 輸出入： `current-input-port`, `current-output-port`, `current-error-port`
- 模組路徑與檔案系統路徑的解析函數：

  `current-module-name-resolver`,

  `current-load-relative-directory`, `current-module-declare-name`
- 資源管理： *custodians*, `current-custodian`, `custodian-limit-memory`
- Reflection 的權限管理： `current-inspector`, `current-code-inspector`
- 檔案與網路存取權限設定： *security guard*,

  `current-security-guard`, `make-security-guard`

語言導向程式設計

一種語言解決不了問題的話，就用兩種 (設計對白)