v.8.2

Racket

Essentials

Sites	main download docs git
Community	packages users@ dev@ irc slack twitter
Running	Put `#lang racket "Hello, world!"` in `hello.rkt` and run `racket hello.rkt`

Primitives

Numbers

Literals	integer `1` rational `1/2` complex `1+2i` floating `3.14` double `6.02e+23` hex `#x29` octal `#o32` binary `#b010101`
Arithmetic	`+ - * / quotient remainder modulo add1 sub1 max min round floor ceiling sqrt expt exp log sin` … `atan`
Compare	`= < <= > >=`
Bitwise	`bitwise-ior bitwise-and bitwise-xor bitwise-not arithmetic-shift integer-length`
Format	`number->string string->number real->decimal-string`
Test	`number? complex?` … `exact-nonnegative-integer?` … `zero? positive? negative? even? odd? exact? inexact?`
Misc	`random`
Match Pattern	`(? number? n) 42`

Strings

Literals	`"Racket"` quoting `"a \" approaches!"` unicode `"λx:(μα.α→α).xx"`
Create	`make-string string string-append build-string string-join`
Observe	`string-length string-ref substring string-split in-string`
Modify	`string-downcase string-upcase string-trim`
Test	`string? string=? string<=? string-ci<=?`
Regexp	`#rx"a|b" #rx"^c(a|d)+r$" regexp-quote regexp-match regexp-split regexp-replace regexp-replace*`
Match	`(? string? s) "Banana?"`

Syntax (Beginner)

Basics

Modules	`(module+ main body ...)`
	`(module+ test body ...)`
	`(require mod-path) (provide id)`

S-expressions	`quote '(a b c) quasiquote unquote `(1 2 ,(+ 1 2))`

Procedure Applications	`(fn arg1 arg2)`
	keyword args `(fn arg1 #:key arg2)`
	`(apply fn arg1 (list arg2))`

Procedures	`(lambda (x) x) (λ (x) x)`
	`(λ (x [opt 1]) (+ x opt))`
	`(λ (x #:req key) (+ x key))`
	`(λ (x #:opt [key 1]) (+ x key))`

Binding	`(let ([x 1] [y 2]) (+ x y))` `(let* ([x 1] [x (+ x 1)]) x)`

Conditionals	`(if (zero? x) 0 (/ 1 x))`
	`(cond [(even? x) 0] [(odd? x) 1]`
	`[else "impossible!"])`
	`and or`

Definitions	`(define x 1)` `(define (f y) (+ x y))`
Iteration	`for for/list for*`
Blocks	`begin when unless`
Require Sub-forms	`prefix-in only-in except-in rename-in for-syntax for-label` …
Provide Sub-forms	`all-defined-out all-from-out rename-out` … `contract-out`

Structures

Definition	`(struct dillo (weight color))`
Create	`(define danny (dillo 17.5 'purple))`
Observe	`(dillo? danny) (dillo-weight danny) (dillo-color danny)`
Modify	`(struct-copy dillo danny ([weight 18.0]))`
Match Pattern	`(dillo w c)`

Pattern Matching

Basics	`(match value [pat body] ...)`
Definitions	`(match-define pat value)`
Patterns	`(quote datum) (list lvp ...) (list-no-order pat ...) (vector`

Pattern	

Bytes

Literals	`#"rawbytes\0"`
Create	`make-bytes bytes`
Numbers	`integer->integer-bytes real->floating-point-bytes`
Observe	`bytes-length bytes-ref subbytes in-bytes`
Modify	`bytes-set! bytes-copy! bytes-fill!`
Conversion	`bytes->string/utf-8 string->bytes/utf-8`
Test	`bytes? bytes=?`
Match Pattern	`(? bytes? b) #"0xDEADBEEF"`

Other

Booleans	`#t #f not equal?`
Characters	`#\a #\tab #\λ char? char->integer integer->char char<=? ... char-alphabetic? ...`
Symbols	`'Racket symbol? eq? string->symbol gensym`
Boxes	`box? box unbox set-box! box-cas!`
Procedures	`procedure? apply compose compose1 keyword-apply procedure-rename procedure-arity curry arity-includes?`
Void	`void? void`
Undefined	`undefined`

Data

Lists

Create	`empty list list* build-list for/list`
Observe	`empty? list? pair? length list-ref member count argmin argmax`
Use	`append reverse map andmap ormap foldr in-list`
Modify	`filter remove ... sort take drop split-at partition remove-duplicates shuffle`
Match Pattern	`(list a b c) (list* a b more) (list top more ...)`

Immutable Hash

Create	`hash hasheq`
Observe	`hash? hash-ref hash-has-key? hash-`

	`lvp ...) (struct-id pat ...) (regexp rx-expr pat) (or pat ...) (and pat ...) (? expr pat ...)`

Syntax (Intermediate)

Basics

Mutation	`set!`
Exceptions	`error with-handlers raise exit`
Promises	`promise? delay force`
Continuations	`let/cc let/ec dynamic-wind call-with-continuation-prompt abort-current-continuation call-with-composable-continuation`
Parameters	`make-parameter parameterize`
External Files Needed at Runtime	`define-runtime-path`
Continuation Marks	`continuation-marks with-continuation-mark continuation-mark-set->list`
Multiple Values	`values let-values define-values call-with-values`

Contracts

Basics	`any/c or/c and/c false/c integer-in vector/c listof list/c ...`
Functions	`-> ->* ->i`
Application	`contract-out recontract-out with-contract define/contract`

Iteration

Sequences	`in-range in-naturals in-list in-vector in-port in-lines in-hash in-hash-keys in-hash-values in-directory in-cycle stop-before stop-after in-stream`
Generators	`generator yield in-generator`

Structures

Sub-structures	`(struct 2d (x y)) (struct 3d 2d (z)) (2d-x (3d 1 2 3))`
Mutation	`(struct monster (type [hp #:mutable])) (define healie (monster 'slime 10)) (set-monster-hp! healie 0)`
Transparency	`(struct cash ($ ¢) #:transparent) (struct->vector (cash 5 95))`

	count in-hash in-hash-keys in-hash-values
Modify	hash-set hash-update hash-remove

Vector

Create	build-vector vector make-vector list->vector
Observe	vector? vector-length vector-ref in-vector
Modify	vector-set! vector-fill! vector-copy! vector-map!
Match Pattern	(vector *x y z*) (vector *x y calabi-yau ...*)

Streams

Create	stream stream* empty-stream
Observe	stream-empty? stream-first stream-rest in-stream

Mutable Hash

Create	make-hash make-hasheq
Observe	hash? hash-ref hash-has-key? hash-count in-hash in-hash-keys in-hash-values
Modify	hash-set! hash-ref! hash-update! hash-remove!

Systems

Input/Output

Formatting	~a ~v ~s ~e ~r pretty-format
Input	read read-bytes peek-byte
Output	write write-bytes display displayln pretty-print
Ports and Files	with-input-from-file with-output-to-file flush-output file-position make-pipe with-output-to-string with-input-from-string port->bytes port->lines ...

Files

Paths	build-path bytes->path path->bytes path-replace-suffix ...
Files	file-exists? rename-file-or-directory copy-directory/files current-directory make-directory delete-directory/files directory-list filesystem-change-evt file->bytes file->lines make-temporary-file

Miscellaneous

Time	current-seconds current-inexact-milliseconds date->string date-

Printing	(struct nickname [n v] #:methods gen:custom-write [(define (write-proc nn p mode) (fprintf p (nickname-n nn)))]) (displayln (nickname "evens" (in-range 0 100 2)))
Serialization	(struct txn (who what where) #:prefab) (write (txn "Mustard" "Spatula" "Observatory"))

Generics

Definition	define-generics
Instantiation	(struct even-set () #:methods gen:set [(define (set-member? st i) (even? i))])

Classes

Definition	interface class*
Instantiation	make-object new instantiate
Methods	send send/apply send/keyword-apply send* send+
Fields	get-field set-field!
Mixins	mixin
Traits	trait trait-sum trait-exclude trait-rename ...
Contracts	class/c instanceof/c is-a?/c implementation?/c subclass?/c

Syntactic Abstractions

Definition	define-syntax define-simple-macro begin-for-syntax for-syntax
Templates	syntax syntax/loc with-syntax
Parsing ()-Syntax	syntax-parse define-syntax-class pattern
Syntax Objects	syntax-source syntax-line ... syntax->datum datum->syntax generate-temporaries format-id
Transformers	make-set!-transformer make-rename-transformer local-expand syntax-local-value syntax-local-name syntax-local-lift-expression ...
Syntax Parameters	define-syntax-parameter syntax-parameterize syntax-parameter-value
Parsing Raw Syntax	lexer parser cfg-parser

	`display-format`
Command-Line Parsing	`command-line`
FFI	`ffi-lib _uint32` … `_fun malloc free`

Networking

TCP	`tcp-listen tcp-connect tcp-accept tcp-close`
HTTP	`http-conn http-conn-open! http-conn-send! http-conn-recv! http-conn-sendrecv! http-sendrecv`
URLs	`string->url url->string url-query`
Email	`smtp-send-message imap-connect` …
JSON	`write-json read-json`
XML	`read-xml write-xml write-xexpr`
Databases	`postgresql-connect mysql-connect sqlite3-connect query-exec query-rows prepare start-transaction` …

Security

Custodians	`make-custodian custodian-shutdown-all current-custodian`
Sandboxes	`make-evaluator make-module-evaluator`

Concurrency

Threads	`thread kill-thread thread-wait make-thread-group`
Events	`sync choice-evt wrap-evt handle-evt alarm-evt` …
Channels	`make-channel channel-get channel-put`
Semaphores	`make-semaphore semaphore-post semaphore-wait`
Async Channels	`make-async-channel async-channel-get async-channel-put`

Parallelism

Futures	`future touch processor-count make-fsemaphore` …
Places	`dynamic-place place place-wait place-wait place-channel` …
Processes	`subprocess system*`

Tools

Packages

Inspection	`raco pkg show`
Finding	pkgs.racket-lang.org
Installing	`raco pkg install`
Updating	`raco pkg update`
Removing	`raco pkg remove`

Miscellaneous

Compiling	`raco make program.rkt`
Testing	`raco test program.rkt a-directory`
Building Executables	`raco exe program.rkt`
Extending DrRacket	`drracket:language:simple-module-based-language->module-based-language-mixin`
Slides	`slide standard-fish code`