CS135 - Constructs Cheat Sheet

When to use • Inputs • Outputs • Examples

Syntax & Meta

Construct	When to use	Inputs	Output	Example
“(...)”	Call function / form	operator + args	value	(add1 5) ⟶ 6
“[...]”	Readable cond blocks	—	—	(cond [(p) 1] [else 0])
“‘ (quote)“	Literal data	datum	same datum	‘(1 2 3)
“;”	Comment	text	—	; note
“define”	Bind value/function	id / (id args ...)	binding	(define (f x) (+ x 1))
“cond / else”	Multi-way branch	tests	value	(cond [(< n 0) −1] [else 1])
“local”	Local helpers	defs	value	(local [(define k 3)] (+ k 2))

Booleans & Logic

Construct	When to use	Inputs	Output	Example
“true / false”	Boolean literals	—	Bool	true
“and / or”	Short-circuit logic	Bool ...	Bool	(and (number? x) (>= x 0))
“not”	Negation	Bool	Bool	(not (empty? xs))

Numbers & Compare

Construct	When to use	Inputs	Output	Example
“+ - * /”	Arithmetic	Num ...	Num	(+ 3 4) ⟶ 7
“= < > <= >=”	Compare numbers	Num Num	Bool	(<= 3 5) ⟶ true
“add1 / sub1”	±1 convenience	Int	Int	(sub1 10) ⟶ 9
“abs”	Absolute value	Num	Num	(abs −5) ⟶ 5
“max / min”	Extremes	Num ...	Num	(max 3 9 4) ⟶ 9
“even? / odd?”	Parity checks	Int	Bool	(odd? 7) ⟶ true
“zero?”	Zero test	Num	Bool	(zero? 0.0) ⟶ true

“quotient”	Integer division	Int Int	Int	(quotient 7 3) ⟶ 2
“remainder”	Remainder	Int Int	Int	(remainder 7 3) ⟶ 1
“exp / log”	e^x and ln	Num	Num	(log (exp 2)) ⟶ 2
“expt / sqr / sqrt”	x^y, x^2, √x	Num	Num	(expt 2 5) ⟶ 32
“pi”	π constant	—	Num	(* 2 pi) ⟶ 6.283…

Trig

Construct	When to use	Inputs	Output	Example
“sin / cos / tan”	Trig (radians)	Num	Num	(sin pi) ⟶ 0
“asin / acos / atan”	Inverse trig	Num	Num	(acos 1) ⟶ 0

Predicates (Type Tests)

Construct	When to use	Input	Output	Example
“number? / integer? / rational? / inexact?”	Numeric kinds	Any	Bool	(integer? 3.0) ⟶ false
“string? / char?”	String/char type	Any	Bool	(string? “hi”) ⟶ true
“symbol?”	Symbol type	Any	Bool	(symbol? ‘rock) ⟶ true
“list? / empty? / cons?”	List shape	Any	Bool	(empty? ‘()) ⟶ true

Equality & Ordering (By Type)

Construct	When to use	Inputs	Output	Example
“=”	Numeric equality	Num Num	Bool	(= 3 3.0) ⟶ true
“char=? / char<?”	Char compare	Char Char	Bool	(char<? #a #b) ⟶ true
“string=? / string<?”	String compare	Str Str	Bool	(string=? “a” “a”) ⟶ true
“symbol=?”	Symbol equality	Sym Sym	Bool	(symbol=? ‘a ‘a) ⟶ true

Lists (Build & Access)

Construct	When to use	Inputs	Output	Example
“empty”	Empty list	—	List	empty
“cons”	Add head	X, (listof X)	(listof X)	(cons 1 ‘(2 3)) ⟶ ‘(1 2 3)
“first / second / third”	Nth selectors	non-empty list	X	(second ‘(9 8 7)) ⟶ 8
“rest”	Tail	non-empty list	list	(rest ‘(1 2 3)) ⟶ ‘(2 3)
“list”	Build list	X …	(listof X)	(list ‘a ‘b 3)
“append”	Concat lists	(listof X) …	(listof X)	(append ‘(1 2) ‘(3)) ⟶ ‘(1 2 3)
“reverse”	Reverse list	(listof X)	(listof X)	(reverse ‘(1 2)) ⟶ ‘(2 1)

Strings & Chars

Construct	When to use	Inputs	Output	Example
“string=? / string<?”	Compare strings	Str Str	Bool	(string<? “a” “b”) ⟶ true
“string->list / list->string”	Convert	Str / (listof Char)	list / Str	(string->list “hi”) ⟶ ‘(#h #i)

Testing

Construct	When to use	Inputs	Output	Example
“check-expect”	Exact expected	actual, expected	test	(check-expect (add1 4) 5)
“check-within”	Approximate (ε)	actual, expected, ε	test	(check-within (sin pi) 0 1e-9)

Contracts & Data (HtDP/CS135)

Name	Meaning / requirement	Example
“Bool”	true or false	(and true false) ⟶ false
“Int / Nat / Num / Rat”	Integer; natural (≥0); any number; rational	Use Nat for sizes/indices

“Str / Char / Sym”	String, character, symbol	‘rock, #a, “hello”
“Atom”	Atomic (non-list) value	number, boolean, symbol, char, string
“Any”	Any value	(Any -> Bool)
“anyof”	Union type	(anyof Int Str)
“listof”	Homogeneous list	(listof Nat) e.g., ‘(1 2 3)

Tiny Patterns

```
;; Branch with cond
(define (signum n)
  (cond [(< n 0) -1]
        [(= n 0) 0]
        [else 1]))

;; Safe list recursion
(define (sum lst)
  (cond [(empty? lst) 0]
        [else (+ (first lst) (sum (rest lst)))]))

;; Type-directed equality
(symbol=? 'rock 'rock)   ; #t
(string=? "a" "a")       ; #t
(= 3 3.0)                ; #t
```