



RASCAL
www.rascal-mpl.org

Cheat Sheet

Module
<pre> module Example import N; data Day = mon() tue(); public int double(int n) { return 2 * n; } private str S = "rascal"; alias Money = int; </pre>

Declarations	
<code>import N</code>	Import module N
<code>data N = P ...</code>	Data type N
<code>T N(T V, ...) S</code> <code>T N(T V, ...) S</code> <code>throws N, N</code>	Function N
<code>T V = E</code>	Variable V
<code>alias N = T</code>	Alias N
<code>anno T T @ N</code>	Annotation N
<code>rule N P => P</code> <code>rule N P : S</code>	Rewrite rule N

Control Statements
<code>if(E) S</code>
<code>if(E) S else S</code>
<code>while(E) S</code>
<code>do S while(E)</code>
<code>for(E,E,...) S</code>
<pre> switch(E) { case P => P case P : S ... default: S } </pre>
<pre> try S catch P => P catch P : S finally: S </pre>
<code>throw E</code>
<code>fail</code>
<code>return, return E</code>
<code>solve(V, V, ...) S</code>

Other statements
<code>{ S; S; ... }</code>
<code>assert E</code> <code>assert E : E</code>
<code>test E</code> <code>test E : E</code>
<code>append E</code>
<code>insert E</code>

Types and Values	
<code>bool</code>	<code>true, false</code>
<code>int</code>	<code>1, 0, -1, 123456789</code>
<code>real</code>	<code>2.30E-14</code>
<code>str</code>	<code>"rascal"</code>
<code>loc</code>	<code> file:///etc/passwd </code>
<code>tuple[T, ...]</code> <code>tuple[T V, ...]</code>	<code><"monday", 1></code>
<code>list[T]</code>	<code>[1,2,3,2,1]</code>
<code>set[T]</code>	<code>{1,2,3}</code>
<code>map[T,T]</code>	<code>("mo":1, "tue": 2)</code>
<code>rel[T, ...]</code> <code>rel[T V, ...]</code>	<code>{<2001,5>, <2002, 6>}</code>
<code>node</code>	<code>f(), g("abc",[2,3,4])</code>
<code>void, value</code>	

Assignment	
<code>V = E</code>	Variable
<code>A[E] = E</code>	Subscript
<code>A.N = E</code>	Field
<code><A,A,...> = E</code>	Tuple
<code>A ? E = E</code>	Isdefined
<code>A @ N = E</code>	Annotation
<code>N(A,A,...) = E</code>	Constructor
<code>+=, -=, *=, /= &=, ?=</code>	<code>A = A op E</code>

Expressions	
<code>E . N</code>	Field selection: <code>x.a</code>
<code>E [N = E]</code>	Field assignment: <code>x[a=3]</code>
<code>E < N, ... ></code>	Field projection <code>x<a></code>
<code>E[E, ...]</code>	Subscription: <code>L["a"]</code>
<code>E @ N</code>	Annotation
<code>E[@N = E]</code>	Annotation replacement
<code>N(E,...)</code>	Function call: <code>f(3, "a")</code>
<code>E ? E : E</code>	Conditional expression: <code>(x > 3) ? 30 : 40</code>
<code>[E .. E]</code> <code>[E, E, .. E]</code>	Range: <code>[1 .. 10]</code> <code>[1, 2 .. 10]</code>
<code>P <- E</code>	Enumerator: <code>n <- [1..10]</code>
<code>[E, ... E, ...]</code>	List comprehension: <code>[n * n int n <- [1 .. 10]]</code>
<code>{E, ... E, ...}</code>	Set comprehension <code>{ n int n <- [1 .. 10], n%3 == 0 }</code>
<code>(E : E E, ...)</code>	Map comprehension <code>(k : size(k) k <- ["a", "bcd"])</code>
<code>visit(E) {</code> <code>case P => P</code> <code>case P : S</code> <code>...</code> <code>}</code>	Visit (traversal: <code>visit(T){</code> <code>case int n => n+1;</code> <code>}</code>
<code>one(E,...)</code>	One E is true
<code>all(E,...)</code>	All Es are true
<code>if, while, do, for</code>	

Operators	
<code>E + E</code>	Addition, union, concatenation
<code>E - E</code>	Subtraction, difference
<code>E * E</code>	Multiplication, product
<code>E / E</code>	Division
<code>E % E</code>	Modulo
<code>E & E</code>	Intersection
<code>E join E</code>	Join
<code>E o E</code>	Compose
<code>E && E</code>	And
<code>E E</code>	Or
<code>E == E, E != E</code>	Equal, Not equal
<code>E < E, E <= E, E > E, E >= E</code>	Comparison
<code>E ==> E</code> <code>E <==> E</code>	Implies Equivalence
<code>E in E</code> <code>E notin E</code>	Element of Not element of
<code>P := E</code> <code>P !:= E</code>	Match No match
<code>- E</code> <code>! E</code>	Arith. Negation, Logical Not
<code>E +, E *</code>	(Reflexive) transitive closure
<code>E ?</code>	Is defined

Abstract Patterns	
$T\ V$	Variable declaration
V^*	Multi-variable (list or set) declaration
V	Variable use
$[P, P, \dots]$	List
$\{P, P, \dots\}$	Set
$\langle P, P, \dots \rangle$	Tuple
$N(P, P, \dots)$	Node
$/ P$	Descendant
$V : P$	Labeled
$T\ V : P$	Typed, Labelled
$[T]\ P$	Type constraint

Concrete Patterns	
$\texttt{' L L ...'}$	Quoted
$T\ \texttt{' L L ...'}$	Typed & Quoted
$L\ L\ \dots$	Unquoted
$\langle T\ V \rangle$	Typed variable

Regular Expression Patterns	
$/ \dots /$	Regular Expression
$\langle V : R \rangle$	Named RegExp
$L\ L\ \dots$	Unquoted
$\langle T\ V \rangle$	Typed variable

Standard Library	
ATermIO	Read/write values as ATerms
Benchmark	Benchmarking tools
Boolean	Boolean functions
Exception	Exceptions a program can catch
Graph	Graph manipulation
Integer	Integer functions
IO	Input/output
JDT	Java fact extraction functions
LabeledGraph	Labeled graph manipulation
List	List functions
Location	Location functions
Map	Map functions
Node	Node functions
Number	Functions on integers and reals

Standard Library	
PriorityQueue	Functions on prio queues
Relation	Functions on relations
Resource	Retrieve Eclipse resources
RSF	Read RSF files
Set	Functions on sets
String	Functions on strings
Tree	Functions on parse trees
Tuples	Functions on tuples
ValueIO	Read/write values as text/binary
viz::Basic	Display basic datatypes
viz::Figure::Chart	Chart drawing
viz::Figure::Core	Core figure functions
viz::Figure::Render	Rendering of figure
viz::View	Graph/tree/text display of values

Legend	
E	Expression
S	Statement
V	Variable
T	Type
A	Assignable
P	Pattern
N	Name
R	Regular Expression
L	Lexical token