



**RASCAL**  
[www.rascal-mpl.org](http://www.rascal-mpl.org)

# Cheat Sheet

## Module

```
module Example
import N;
data Day = mon() | tue();
public int twice(int n) {
    return 2 * n;
}
private str S = "rascal";
alias Money = int;
```

## Declarations

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

## Control Statements

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

## Other statements

```
{ S; S; ... }
assert E
assert E : E
test E
test E : E
append E
insert E
```

## Types and Values

<code>bool</code>	<code>true, false</code>
<code>int, real, num</code>	<code>0, -1, 1234, 2.3E-14</code>
<code>str</code>	<code>"rascal"</code>
<code>loc</code>	<code> file:///etc/passwd </code>
<code>datetime</code>	<code>\$2010-07-15</code>
<code>tuple[T, ...]</code> <code>tuple[T V, ...]</code>	<code>&lt;"monday", 1&gt;</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>{&lt;2001,5&gt;, &lt;2002, 6&gt;}</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>&lt;A,A,...&gt; = E</code>	Tuple
<code>A ? E = E</code>	Isdefined
<code>A @ N = E</code>	Annotation
<code>N(A,A,...) = E</code>	Constructor
<code>+=, -=, *=, /= &amp;=, ?=</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 &lt; N, ... &gt;</code>	Field projection <code>x&lt;a&gt;</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 &gt; 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 &lt;- E</code>	Enumerator: <code>n &lt;- [1..10]</code>
<code>[E, ...   E, ...]</code>	List comprehension: <code>[ n * n   int n &lt;- [1 .. 10]]</code>
<code>{E, ...   E, ...}</code>	Set comprehension <code>{ n   int n &lt;- [1 .. 10], n%3 == 0 }</code>
<code>(E : E   E, ...)</code>	Map comprehension <code>(k : size(k)   k &lt;- ["a", "bcd"])</code>
<code>visit(E) {</code> <code>case P =&gt; P</code> <code>case P : S</code> <code>...</code> <code>}</code>	Visit (traversal: <code>visit(T){</code> <code>case int n =&gt; n+1;</code> <code>}</code>
<code>one(E,...)</code>	One <i>E</i> is true
<code>all(E,...)</code>	All <i>E</i> s 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 &amp; E</code>	Intersection
<code>E join E</code>	Join
<code>E o E</code>	Compose
<code>E &amp;&amp; E</code>	And
<code>E    E</code>	Or
<code>E == E, E != E</code>	Equal, Not equal
<code>E &lt; E, E &lt;= E, E &gt; E, E &gt;= E</code>	Comparison
<code>E ==&gt; E</code> <code>E &lt;==&gt; 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

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

Standard Library	
<a href="#">ATermIO</a>	Read/write values as ATerms
<a href="#">Benchmark</a>	Benchmarking tools
<a href="#">Boolean</a>	Boolean functions
<a href="#">Exception</a>	Exceptions a program can catch
<a href="#">Graph</a>	Graph manipulation
<a href="#">Integer</a>	Integer functions
<a href="#">IO</a>	Input/output
<a href="#">Java</a>	Java fact model
<a href="#">JDT</a>	Java fact extraction functions
<a href="#">LabeledGraph</a>	Labeled graph manipulation
<a href="#">List</a>	List functions
<a href="#">Location</a>	Location functions
<a href="#">Map</a>	Map functions
<a href="#">Node</a>	Node functions

Standard Library	
<a href="#">Number</a>	Functions on integers and reals
<a href="#">PriorityQueue</a>	Functions on priority queues
<a href="#">Relation</a>	Functions on relations
<a href="#">Resources</a>	Retrieve Eclipse resources
<a href="#">RSF</a>	Read RSF files
<a href="#">Scripting</a>	Evaluate Rascal expressions
<a href="#">Set</a>	Functions on sets
<a href="#">String</a>	Functions on strings
<a href="#">Tree</a>	Functions on parse trees
<a href="#">Tuples</a>	Functions on tuples
<a href="#">ValueIO</a>	Read/write values as text/binary
<a href="#">vis::Chart</a>	Chart drawing
<a href="#">vis::Figure</a>	Core figure functions
<a href="#">vis::Render</a>	Rendering of figure