# Rascal Cheat Sheet

www.rascal-mpl.org 2-10-2010

## 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 | |
|---|---|
| import *N* | Import module *N* |
| data *N* = *P* \| ... | Data type *N* |
| *T N*(*T V*, ...) *S*<br>*T N*(*T V*, ...) *S*<br>throws *N*, *N* | Function *N* |
| *T V* = *E* | Variable *V* |
| alias *N* = *T* | Alias *N* |
| anno *T T* @ *N* | Annotation *N* |
| rule *N P* => *P*<br>rule *N P* : *S* | Rewrite rule *N* |

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

| Other statements |
|---|
| { *S*; *S*; ... } |
| assert *E*<br>assert *E* : *E* |
| test *E*<br>test *E* : *E* |
| append *E* |
| insert *E* |

| Types | and Values |
|---|---|
| bool | true, false |
| int, real, num | 0, -1, 1234, 2.3E-14 |
| str | "rascal" |
| loc | \|file:///etc/passwd\| |
| datetime | $2010-07-15 |
| tuple[*T*, ...]<br>tuple[*T V*, ...] | <"monday", 1> |
| list[*T*] | [1,2,3,2,1] |
| set[*T*] | {1,2,3} |
| map[*T*,*T*] | ("mo":1, "tue": 2) |
| rel[*T*, ...]<br>rel[*T V*, ...] | {<2001,5>, <2002, 6>} |
| node | f(), g("abc",[2,3,4]) |
| void, value | |

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

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

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

| Abstract Patterns | |
|---|---|
| *T V* | Variable declaration |
| *V** | Multi-variable (list or set) declaration |
| *V* | Variable use |
| [*P*, *P*, ...] | List |
| {*P*, *P*, ...} | Set |
| ‹*P*, *P*, ...› | Tuple |
| *N*(*P*, *P*, ...) | Node |
| / *P* | Descendant |
| *V* : *P* | Labeled |
| *T V* : *P* | Typed, Labelled |
| [*T*] *P* | Type constraint |

| Concrete Patterns | |
|---|---|
| ` *L L* ...` | Quoted |
| *T* ` *L L* ...` | Typed & Quoted |
| *L L* ... | Unquoted |
| ‹*T V*› | Typed variable |

| Regular Expression Patterns | |
|---|---|
| / ... / | Regular Expression |
| ‹*V* : *R*› | Named RegExp |
| *L L* ... | Unquoted |
| ‹*T V*› | 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 | |
|---|---|
| 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 |
| Java | Java fact model |
| JDT | Java fact extraction functions |
| LabeledGraph | Labeled graph manipulation |
| List | List functions |
| Location | Location functions |
| Map | Map functions |
| Node | Node functions |

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