Mercury Programming Language Cheatsheet

A concise reference for the Mercury programming language, covering syntax, data types, control flow, and module system.

Core Syntax and Types

Basic Syntax

Comments	`% This is a single-line comment` `/* This is a multi-line comment */`
Predicates	`pred my_predicate(in Arg1, out Arg2) is det.`
Functions	`func my_function(Arg1) = Result.`
Clauses	`my_predicate(A, B) :- A > 0, B = A * 2.`
Mode Declarations	`mode my_mode == in(int), out(int).`
Type Declarations	`:- type color ---> red ; green ; blue.`

Data Types

Integer	`int`
Float	`float`
String	`string`
Boolean	`bool`
List	`list(T)`
Tuple	`{T1, T2, ..., Tn}`
User-defined Types	Using `:- type` declarations

Mode Annotations

Mode annotations specify how arguments are used in a predicate or function. Common modes include:

- `in`: Input argument.
- `out`: Output argument.
- `in_out`: Argument is both input and output.

Control Flow and Logic

Conditional Statements

If-Then-Else

```
if Condition then
  Action1
else
  Action2
end if.
```

If-Then

```
if Condition then
  Action
end if.
```

Loops and Recursion

Recursion

Mercury encourages recursion for iterative processes.

```
pred process_list(in list(T), out list(Result)) is det.
process_list([], []).
process_list([H | T], [R | Result]) :-
  process_element(H, R),
  process_list(T, Result).
```

Determinism

Mercury uses determinism annotations to indicate the number of solutions a predicate can produce:

- `det`: Exactly one solution.
- `semidet`: Zero or one solution.
- `multi`: Zero or more solutions.
- `failure`: No solution.
- `cc_multi`: Coroutining multi.

Negation

Negation as Failure

```
not(Predicate) :-
  Predicate ->
    fail
  ;
    true.
```

Module System

Module Declaration

Module Header	`:- module my_module.`
Interface Declaration	`:- interface.`
Implementation Declaration	`:- implementation.`

Importing Modules

Importing Predicates	`:- import_module module_name.`
Selective Import	`:- import_module module_name([predicate1, predicate2]).`

Exporting Symbols

Exporting Predicates

```
:- pred my_predicate(in Arg1, out Arg2) is det.
:- export my_predicate/2.
```

Error Handling and Debugging

Exception Handling

Throwing Exceptions	`:- exception my_exception(string).`
	
	`raise_exception :-`
	`  throw(my_exception("An error occurred")).`

Catching Exceptions	`try(`
	`  Goal,`
	`  Catcher,`
	`  Recoverer`
	`).`

Debugging

Mercury provides debugging tools for tracing predicate execution and inspecting variables. Utilize the Mercury debugger (if available in your environment) to step through code, set breakpoints, and examine the state of variables.