



Literate programming

<code># Title</code>	Heading
<code>### Sub-subtitle</code>	
<code>```catala ```catala-metadata</code> <code>```</code>	Code / metadata block
<code>> Module Mdl</code>	Module declaration
<code>> Using Mdl as M</code>	Module import+alias
<code>> Include: foo.catala_en</code>	File inclusion
<code>```catala-test-cli</code> <code>\$ catala interpret --scope Scope1</code> <code>```</code>	Command-line interface test

Literals and types

<code>true</code>	<code>false</code>	boolean
<code>65,536</code>		integer
<code>65536.262144</code>	<code>37%</code>	decimal
<code>\$1,234,567.89</code>		money
<code> 2026-05-11 </code>		date
<code>254 day -4 month 1 year</code>		duration
<code>Case1 content 12 Case2</code>		Enum1
<code>Struct1 { -- fld1: 9 -- fld2: 7% }</code>		Struct1
<code>Present content \$34 Absent</code>		optional of money
<code>[12; 24; 36]</code>		list of integer
<code>[]</code>		list of anything of type t
<code>(2012-02-03 , \$30, 1%)</code>		(date,money,decimal)
<code>f of x, y equals</code> <code> y * x / \$12.0</code>		decimal depends on x content money, y content decimal

Operators and built-ins

<code>not a</code>	<code>a and b</code>	
<code>a or b</code>	<code># "or otherwise"</code>	Logical operators
<code>a xor b</code>	<code># exclusive or</code>	
<code>- a</code>	<code>a + b</code>	<code>a - b</code>
<code>a * b</code>	<code>a / b</code>	
<code>a = b</code>	<code>a < b</code>	<code>a <= b</code>
<code>a != b</code>	<code>a > b</code>	<code>a >= b</code>
<code>decimal of 44</code>		Conversions
<code>money of 23.15</code>		
<code>round of \$9.99</code>		Rounding
<code>Date.get_month of ...</code>		Date parts
<code>Date.first_day_of_month of ...</code>		

Metadata declaration

<code>declaration structure Struct1:</code> <code> data fld1 content integer</code> <code> data fld2 content decimal</code>	Structure declaration
<code>declaration enumeration Enum1:</code> <code> -- Case1 content integer</code> <code> -- Case2</code>	Enumeration declaration
<code>## Documentation for Scope1</code> <code>#[test]</code>	Documentation string Test scope annotation
<code>declaration scope Scope1:</code> <code> internal var1 content integer</code> <code> state before</code> <code> state after</code> <code> internal var2 condition</code> <code> sub1 scope Scope0</code> <code> output var3 content ...</code> <code> input var4 content ...</code> <code> input output var5 content ...</code> <code> context var6 content ...</code> <code> context output var7 content ...</code> <code> output sub2 scope Scope0</code>	Scope declaration Scope variable State transitions Condition Sub-scope declaration
<code>declaration const content decimal</code> <code> equals 17.1</code>	Global definition
<code>declaration square content decimal</code> <code> depends on x content decimal</code> <code> equals x * x</code>	Global function definition

Expressions

<code>let x equals 36 - 5 in ...</code>	Local definition
<code>match expr with pattern</code> <code> -- Case1 content x : ...</code> <code> -- Case2 : ...</code> <code> -- anything : ...</code>	Pattern matching
<code>#[error.message = "err"] impossible</code>	Unreachable code
<code>if ... then ... else ...</code>	Conditional
<code>#[debug.print = "message"] expr</code>	Debug annotation
<code>expr with pattern Case1</code> <code> content x and x >= 2</code>	Pattern test (optional: var. binding)
<code>struc1 but replace { -- fld2: 8% }</code>	Field replacement
<code>struc1.fld2 tuple1.2</code> <code>sub1.var0</code>	Field, tuple element, subscope variable
<code>f of \$44.50, 1/3</code>	Function call
<code>output of Scope1 with</code> <code> { -- fld1: 9 -- fld2: 15% }</code>	Direct scope call
<code>var1 state before</code>	Variable state access
<code>assertion x > 0 in ...</code>	Local assertion

Scope definition

<code>scope Scope1: ...</code>	Defining scope contents
<code>scope Scope1</code> <code> under condition var1 >= 2: ...</code>	Use-wide condition
<code>definition var1 equals ...</code>	Unconditional def.
<code>definition var1</code> <code> under condition ...</code> <code> consequence equals ...</code>	Conditional definition
<code>rule var2</code> <code> under condition var1 >= 2</code> <code> consequence not fulfilled</code>	Rule (definition for conditions)
<code>definition f of x, y equals ...</code>	Function def. or rule
<code>label lbl1 definition var1 ...</code>	Labeled def. or rule
<code>exception lbl1 definition var1 ...</code>	Exception to label
<code>exception definition var1 ...</code>	Exception to implicit
<code>definition var1 state before</code> <code> equals ...</code>	State definition
<code>assertion var1 > 0</code>	Scope-level assertion
<code>date round down up</code>	Date rounding mode

List operations

<code>lst contains 3</code>	Presence test
<code>exists x among lst such that x > 2</code>	Existence test
<code>for all x among lst we have x > 2</code>	For all test
<code>map each x among lst to x + 2</code>	Mapping
<code>list of x among lst such that x > 2</code>	Filter
<code>map each x among lst such that x > 2</code> <code> to x - 2</code>	Filter + map
<code>map each (x, y) among (lst1, lst2)</code> <code> to x + y</code>	Multiple mapping
<code>lst1 ++ lst2</code>	Merge
<code>number of lst</code>	Count
<code>maximum of lst</code> <code> or if list empty then -1</code>	Extremum search (optional default)
<code>content of x among lst</code> <code> such that x.fld1 = ...</code>	Search by criterion
<code>content of x among lst</code> <code> such that x * x is minimum</code> <code> or if list empty then -1</code>	Arg-extremum search (optional default)
<code>sort lst in in-decreasing order</code>	Sorting
<code>sort all x among lst</code> <code> in in-decreasing order of x.fld2</code> <code> and then -x.fld1 ...</code>	Sorting on criteria (optional sub-criteria)
<code>combine all x among lst</code> <code> in acc initially 0</code> <code> with acc + x</code>	Fold (loop and accumulate)

LA SYNTAXE DE CATALA français

Programmation littéraire

```
# Titre
### Sous-sous-titre
# Article 1 | JORFARTI000012345678
# Article 2 | LEGIARTI000012345678
# Décision 3 | CETATEXT000012345678
```catala      ```catala-metadata
```
> Module Mdl
> Usage de Mdl en tant que M
> Inclusion: foo.catala_en
```catala-test-cli
$ catala interpret --scope Chp1
```
```

En-têtes

Référence au journal officiel

Bloc de code / métadonnées

Déclaration de module

Import de module

Inclusion textuelle

Test intégré

Littéraires et types

```
vrai          faux
65536
65536,262144  37%
1 234 567,89€
|2026-05-11|
254 jour    -4 mois    1 an
Cas1 contenu 12      Cas2
Struct1 { -- chp1: 9 -- chp2: 7% }
Présent contenu 34€ Absent
[ 12; 24; 36 ]
```

booléen

entier

décimal

argent

date

durée

Énum1

Struct1

optionnel de argent

liste de entier

liste de

n'importe quel de type t

(date, argent, décimal)

décimal dépend de

x contenu argent, y contenu décimal

Opérations

```
non a          a et b
a ou b         # "ou à défaut"
a ou bien b    # ou exclusif
- a            a + b      a - b
a * b          a / b
a = b          a < b      a <= b
a != b         a > b      a >= b
décimal de 44
argent de 23,15
arrondi de 9,99€
Date.accès_année de ...
Date.premier_jour_du_mois de ...
```

Opérateurs logiques

Arithmétique

Comparaisons

Conversions

Arrondis

Éléments de dates

Déclaration des métadonnées

```
déclaration structure Struct1:
  donnée chp1 contenu entier
  donnée chp2 contenu décimal
déclaration énumération Énum1:
  -- Cas1 contenu entier
  -- Cas2
## Documentation de Chp1
#[test]
déclaration champ d'application Chp1:
  interne var1 contenu entier
  état avant
  état après
  interne var2 condition
  sub1 champ d'application Chp0
  résultat var3 contenu ...
  entrée var4 contenu ...
  entrée résultat var5 contenu ...
  contexte var6 contenu ...
  contexte résultat var7 contenu ...
  résultat sub2
  champ d'application Chp0
déclaration const contenu décimal
  égal à 17,1
déclaration carré contenu décimal
  dépend de x contenu décimal
  égal à x * x
```

Déclaration de structure

Déclaration d'énumération

Texte de documentation

Annot. champ de test

Déclaration de champ

Variable de champ

Transitions d'état

Condition

s/s champ

Qualificateurs d'entrée-sortie

Définition globale

Définition de fonction globale

Expressions

```
soit x égal à 36 - 5 dans ...
selon expr sous forme
-- Cas1 contenu x : ...
-- n'importe quel : ...
#[error.message = "err"] impossible
si ... alors ... sinon ...
#[debug.print = "message"] expr
expr sous forme Cas1
  contenu x et x >= 2
struc1 mais en remplaçant
  { -- chp2: 8% }
struc1.chp2      tuple1.2
sub1.var0
f de 44,50€, 1/3
résultat de Chp1
  avec { -- chp1: 9 -- chp2: 15% }
var1 état avant
assertion x > 0 dans ...
```

Définition locale

Filtrage par motif

Calcul inaccessible

Branchement

Annot. de débog

Test de filtrage (optionnel: test contenu)

Remplacement de champs

Champ, élément de n-uplet, s/s-variable

Appel de fonction

Appel direct de champ d'application

Accès à un état

Assertion locale

Définition de champ d'application

```
champ d'application Chp1: ...
champ d'application Chp1
  sous condition var1 >= 2: ...
définition var1 égal à ...
définition var1
  sous condition ...
  conséquence égal à ...
règle var2
  sous condition var1 >= 2
  conséquence ·non· rempli
définition f de x, y égal à ...
étiquette étq1 définition var1 ...
exception étq1 définition var1 ...
exception définition var1 ...
définition var1 état avant
  égal à ...
assertion var1 > 0
date arrondi inf·supérieur
```

Définition des membres

Avec condition générale

Déf. inconditionnelle

Définition conditionnelle

Règle (définition de condition)

Déf./règle fonction

Déf./règle étiquetée

Exc. à déf. étiquetée

Exception à implicite

Définition d'états

Assertion de champ

Mode arrondi dates

Opérations sur les listes

```
lst contient 3
existe x parmi lst tel que x > 2
pour tout x parmi lst on a x > 2
transforme chaque x parmi lst
  en x - 2
liste de x parmi lst tel que x > 2
transforme chaque x parmi lst
  tel que x > 2 en x - 2
transforme chaque (x, y)
  parmi (lst1, lst2) en x + y
lst1 ++ lst2
nombre de lst
maximum de lst
  ou si liste vide alors -1
contenu de x parmi lst
  tel que x.chp1 = 0
contenu de x parmi lst
  tel que x * x est minimum
  ou si liste vide alors -1
trie lst par ordre décroissant
trie tout x parmi lst
  par ordre décroissant de x.chp2
  puis -x.chp1 ...
combine tout x parmi lst
  dans acc initialement 0
  avec acc + x
```

Test de présence

Test d'existence

Test pour tout

Application un-à-un

Filtrage

Filtrage + application

Application multiple

Réunion

Comptage Recherche d'extremum (optionnel: défaut)

Recherche par critère

Extremum selon critère (optionnel: défaut)

Tri

Tri selon critère

(optionnel: sous-critère)

Fold (boucle avec accumulation)