index

September 15, 2025

Contents

1 index

This module imports all the other modules (directly or indirectly).

They can be reached either using the links in the navigation panels, or by clicking on the module names in the highlighted code below.

```
module index where

import Test
import Test.All
import Test.Sub.Not-Imported
```

2 Test/All

First text line

```
module Test.All where

import Test.Sub.Base

-- Just testing...
```

Literate prose is included in generated webpages and PDFs. It is generally rendered as plain text:

* Blank lines produce paragraph breaks, but line breaks and indentation are otherwise ignored.

* URLs do not become links.

* Characters treated as special by LaTeX lead to errors when generating PDFs.

* Use of Unicode characters in generated PDFs may require mapping them to LaTeX markup.

The subset of LaTeX math markup supported by KaTeX (https://katex.org) is rendered correctly in webpages and PDFs. The following examples are from the Material for MkDocs website:

$$\cos x = \sum_{k=0}^{\infty} \frac{(-1)^k}{(2k)!} x^{2k}$$

The homomorphism f is injective if and only if its kernel is only the singleton set e_G, because otherwise $\exists a, b \in G$ with $a \neq b$ such that $f(a) = f(b)$.

```
-- More code
```

Last text line

3 Test

Copied from examples/syntax/highlighting/Test3.lagda in the agda/agda repository, with additional literate prose by @pdmosses

This test file currently lacks module-related stuff.

And interesting uses of shadowing.

```
module Test where

infix  12 _!
infixl 7 _+_ _-_
infixr 2 -_

data ℕ : Set where
  zero : ℕ
  suc : ℕ -> ℕ
```

The type Set is declared in the built-in module Agda.Primitive.

```
_+_ : ℕ -> ℕ -> ℕ
zero + n = n
suc m + n = suc (m + n)

postulate _-_ : ℕ -> ℕ -> ℕ

-_ : ℕ -> ℕ
- n = n

_! : ℕ -> ℕ
zero ! = suc zero
suc n ! = n - n !

record Equiv {a : Set} (_≈_ : a -> a -> Set) : Set where
  field
    refl  : forall x     -> x ≈ x
    sym : {x y : a} -> x ≈ y -> y ≈ x
    _`trans`_ : forall {x y z} -> x ≈ y -> y ≈ z -> x ≈ z

data _≡_ {a : Set} (x : a) : a -> Set where
  refl : x ≡ x

subst : forall {a x y} ->
  (P : a -> Set) -> x ≡ y -> P x -> P y
subst {x = x} .{y = x} _ refl p = p

Equiv-≡ : forall {a} -> Equiv {a} _≡_
Equiv-≡ {a} =
  record { refl = \_ -> refl
         ; sym = sym
         ; _`trans`_ = _`trans`_
         }
```

```
  where
  sym : {x y : a} -> x ≡ y -> y ≡ x
  sym refl = refl

  _`trans`_ : {x y z : a} -> x ≡ y -> y ≡ z -> x ≡ z
  refl `trans` refl = refl

postulate
  String : Set
  Char   : Set
  Float  : Set

{-# BUILTIN STRING String #-}
{-# BUILTIN CHAR   Char  #-}
{-# BUILTIN FLOAT  Float #-}

{-# BUILTIN NATURAL ℕ  #-}

data [_] (a : Set) : Set where
  []  : [ a ]
  _::_ : a -> [ a ] -> [ a ]

{-# BUILTIN LIST [_] #-}
-- {-# BUILTIN NIL  []  #-}
-- {-# BUILTIN CONS _::_ #-}

primitive
  primStringToList : String -> [ Char ]

string : [ Char ]
string = primStringToList "∃ apa"

char : Char
char = '∀'

anotherString : String
anotherString = "¬ be\
    \pa"

nat : ℕ
nat = 45

float : Float
float = 45.0e-37
```

4 Test/Sub/Base

```
module Test.Sub.Base where
```

This module is imported by Test.All, and should be included in the website generated from Test.All.

5 Test/Sub/Not-Imported

```
module Test.Sub.Not-Imported where

import Test
```

This module is not imported by Test.All, and should not be included in the website generated from Test.All.