



TUTORIAL
Idris

The background features a dark brown gradient with a subtle texture of red wavy lines that curve across the frame, creating a sense of depth and motion.

INTEGRANTES



Diana Valentina **Monroy Molina**

Nicolas Felipe **Pardo Machett's**

Jose Luis **Rativa Medina**

Daniela **Tocua Perilla**

CONTENIDO

03



Primeros Pasos



Sobre el lenguaje

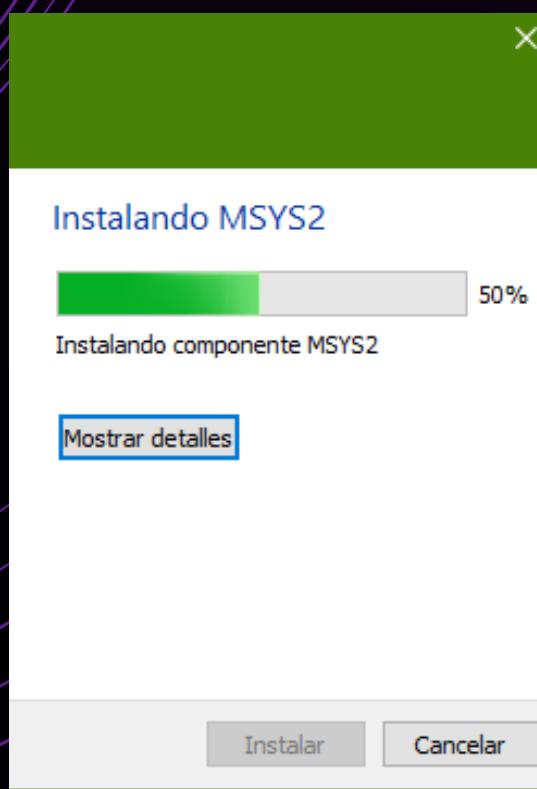


Ejemplos

PRIMEROS PASOS INSTALACION WINDOWS

04

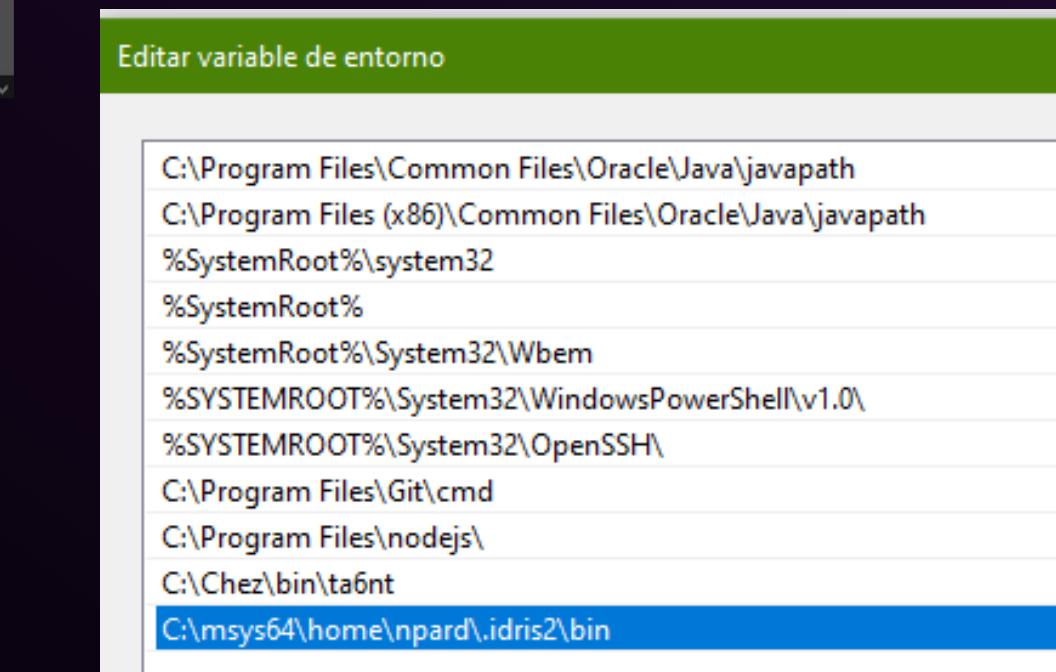
1) Descargar ambiente MSYS2



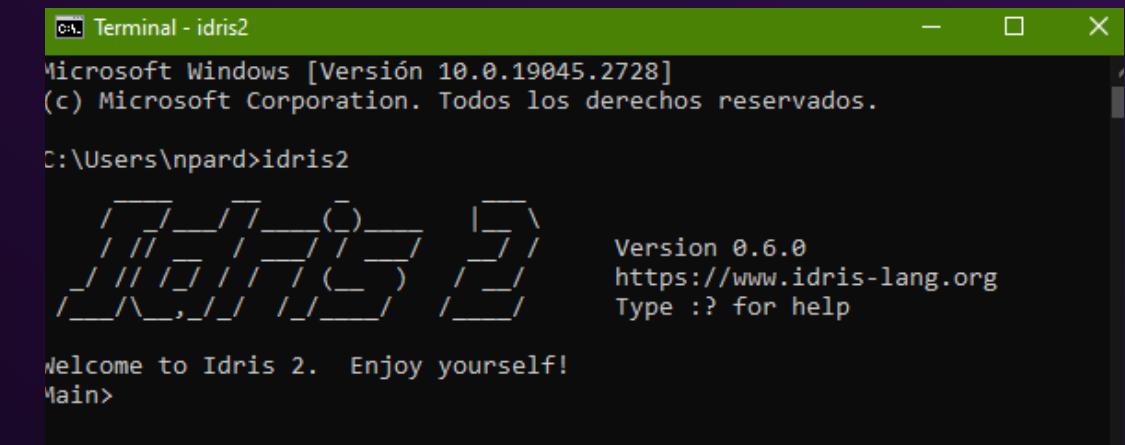
2) Desempaquetar y compilar Idris2

```
M ~
mingw-w64-x86_64-gcc-libs-... 964.4 KiB 774 KiB/s 00:01 [#####] 100%
mingw-w64-x86_64-headers-g... 5.7 MiB 913 KiB/s 00:06 [#####] 100%
mingw-w64-x86_64-libiconv-... 719.8 KiB 629 KiB/s 00:01 [#####] 100%
mingw-w64-x86_64-binutils-... 6.1 MiB 932 KiB/s 00:07 [#####] 100%
mingw-w64-x86_64-zstd-1.5.... 621.6 KiB 752 KiB/s 00:01 [#####] 100%
make-4.4.1-1-x86_64 505.9 KiB 580 KiB/s 00:01 [#####] 100%
mingw-w64-x86_64-gmp-6.2.1... 571.7 KiB 376 KiB/s 00:02 [#####] 100%
mingw-w64-x86_64-mpc-1.3.1... 104.4 KiB 163 KiB/s 00:01 [#####] 100%
mingw-w64-x86_64-zlib-1.2.... 103.8 KiB 160 KiB/s 00:01 [#####] 100%
mingw-w64-x86_64-winpthread... 40.0 KiB 102 KiB/s 00:00 [#####] 100%
mingw-w64-x86_64-libwinpthread... 29.0 KiB 50.8 KiB/s 00:01 [#####] 100%
mingw-w64-x86_64-windows-d... 3.1 KiB 6.55 KiB/s 00:00 [#####] 100%
mingw-w64-x86_64-mpfr-4.2.... 429.4 KiB 171 KiB/s 00:03 [#####] 100%
mingw-w64-x86_64-gcc-13.1.... 28.8 MiB 2036 KiB/s 00:14 [#####] 100%
Total (16/16) 49.4 MiB 3.37 MiB/s 00:15 [#####] 100%
(16/16) checking keys in keyring
(16/16) checking package integrity
(16/16) loading package files
(16/16) checking for file conflicts
(16/16) checking available disk space
:: Processing package changes...
( 1/16) installing make
( 2/16) installing mingw-w64-x86_64-libwinpthread-git
( 3/16) installing mingw-w64-x86_64-gcc-libs
( 4/16) installing mingw-w64-x86_64-zstd
( 5/16) installing mingw-w64-x86_64-binutils
( 6/16) installing mingw-w64-x86_64-headers-git
[#####-----] 6%
```

3) Añadir a PATH



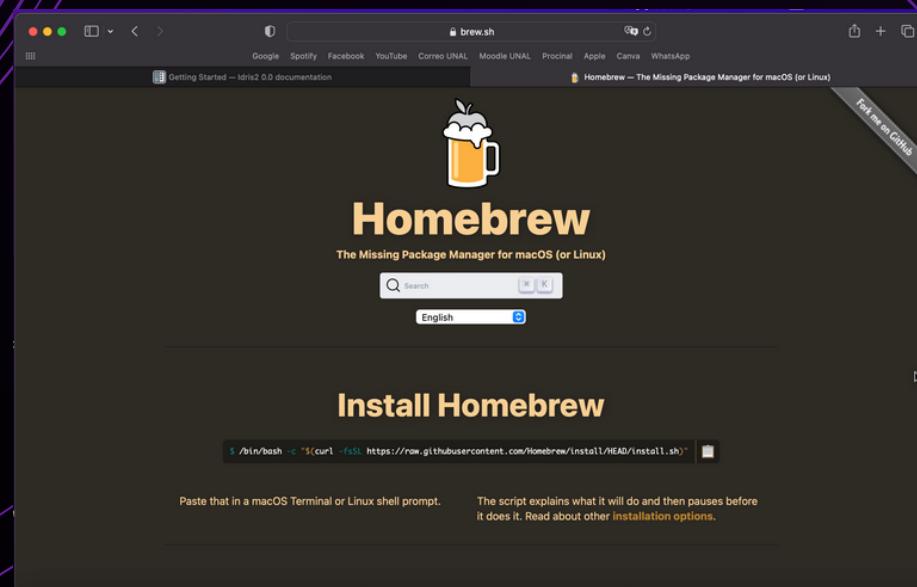
4) Ejecutar



[HTTPS://IDRIS2.READTHEDOCS.IO/EN/LATEST/TUTORIAL/WINDOWS.HTML](https://idris2.readthedocs.io/en/latest/tutorial/windows.html)

PRIMEROS PASOS INSTALACION MAC/LINUX

1) Descargar gestor de paquetes **Homebrew**



2) Instalar Idris2

```
npardom -- zsh -- 63x14
Last login: Fri May 12 21:11:28 on ttys000
npardom@MacBook-Pro-de-Nicolas ~ % brew install idris2
==> Downloading https://formulae.brew.sh/api/formula.jws.json
#####
==> Downloading https://formulae.brew.sh/api/cask.jws.json
#####
Warning: idris2 0.6.0 is already installed and up-to-date.
To reinstall 0.6.0, run:
  brew reinstall idris2
npardom@MacBook-Pro-de-Nicolas ~ %
```

3) Ejecutar

```
npardom -- chez < idris2 -- 80x24
Last login: Sat May 13 10:35:10 on ttys000
npardom@MacBook-Pro-de-Nicolas ~ % idris2
/ _/ / / / / ( ) _/ \ _/ \
/ // _/ / _/ / _/ \ _/ \
/_/ / / / / / ( _ ) / _/ \
/_/ \ _/ / / / _/ / _/ \
Version 0.6.0
https://www.idris-lang.org
Type :? for help

Welcome to Idris 2. Enjoy yourself!
Main>
```

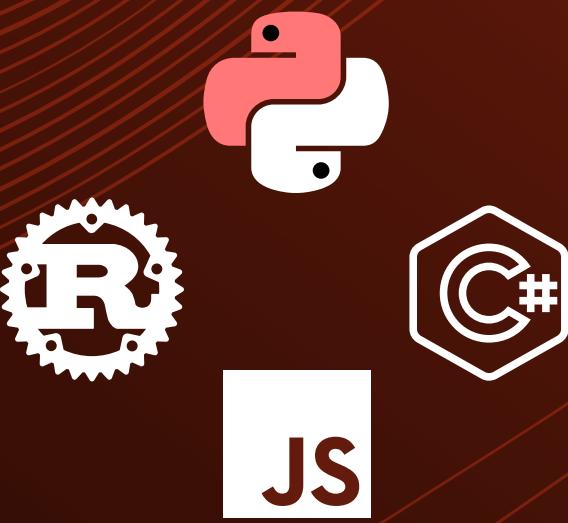
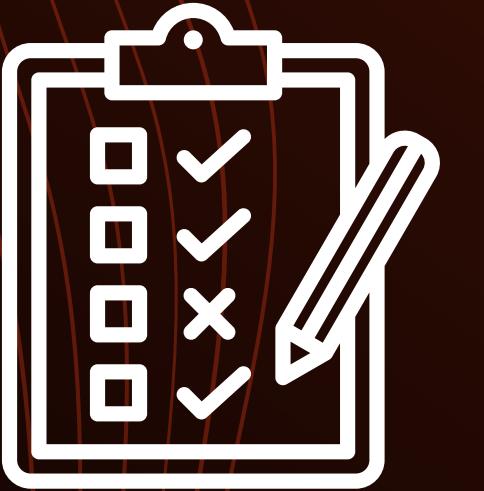
[HTTPS://IDRIS2.READTHEDOCS.IO/EN/LATEST/TUTORIAL/STARTING.HTML](https://idris2.readthedocs.io/en/latest/tutorial/starting.html)

CARACTERÍSTICAS



Lenguaje de
programación
puramente
funcional

Tipos dependientes,
evaluación diferida
opcional



Diseñado para
ser un lenguaje
de **propósito
general**.

CARACTERÍSTICAS



Basado en
Haskell

Tipos de datos de
primer orden



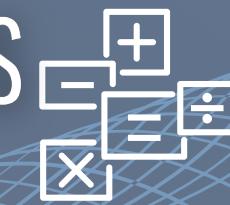
Verificador de
totalidad

SOBRE EL LENGUAJE

TIPOS Y FUNCIONES

DATOS PRIMITIVOS

OPERACIONES NUMERICAS

**Int**Signo de
tamaño fijo**Integer**
Signo de **tamaño dinámico**

1,5

Double

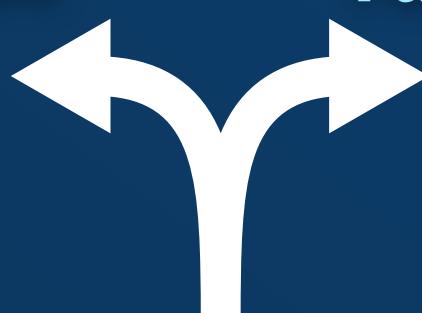
MANIPULACIÓN DE TEXTO

Char**String**

PUNTEROS

**Ptr**Referencia a una
posición de memoria

BOOLEAN

True**False**

EJEMPLO

module Prims**x : Int****x = 42****foo : String****foo = "Sausage machine"****bar : Char****bar = 'Z'****quux : Bool****quux = False**

TIPOS Y FUNCIONES

TIPOS DE DATOS PERSONALIZADOS

Operador **infix**

Operador entre dos operandos

EJEMPLO

3 + 5



OPERADOR INFIX

Creación de un **operador infix**

```
infixr 10 ::
```

Definición del operador :: como infixr (asociativo a la **derecha**) con una **prioridad** de 10.

Operadores **basicos**

:+ - * \ / = . ? | & > < ! @ \$ % ^ ~ #

Operadores **indefinibles**
por el usuario

:	=>	->	<-	=	?=		
	**	==>	\	%	~	?	!

TIPOS Y FUNCIONES

TIPOS DE DATOS PERSONALIZADOS

Declarados como en **Haskell**

```
data Nat = Z | S Nat
```

```
data List a = Nil | (:) a (List a)
```

También se pueden **declarar** dando solo los tipos de los **constructores**

```
data Nat : Type where
```

```
Z : Nat
```

```
S : Nat -> Nat
```

```
data List : Type -> Type where
```

```
Nil : List a
```

```
(::) : a -> List a -> List a
```

TIPOS Y FUNCIONES

FUNCIONES

Se definen utilizando el nombre de la **función**, seguida de los argumentos y su tipo de retorno:

```
add: (a: Int) -> (b: Int) -> Int;  
add a b = a + b;
```

- **No hay restricción** sobre si los nombres de función deben empezar con mayúscula.
- Sin embargo, por convención, **los tipos y los nombres de constructores suelen empezar con mayúscula**.

```
fact : (n : Nat) -> Nat  
fact Z = 1  
fact (S k) = (S k) * fact k
```

Ejemplo de función **recursiva**

TIPOS Y FUNCIONES

FUNCIONES

```
myFunction : (x : Int) -> Int
myFunction x = y * 2
  where
    y : Int
    y = x + 1

main : IO ()
main = putStrLn (show (myFunction 5))
```

- Las funciones también pueden definirse localmente mediante **where**.
- La indentación es **importante**: las funciones del bloque where **deben tener más indentación que la función externa**.

TIPOS Y FUNCIONES

FUNCIONES

14

```
module Main

even : Nat -> Bool
even Z = True
even (S k) = odd k where
  odd : Nat -> Bool
  odd Z = False
  odd (S k) = even k

main : IO ()
main = do
  putStrLn (show (even 44))
  putStrLn (show (even 27))
```

Función para definir si un número es par
(**even** y **odd** se llaman recursivamente)

```
PS C:\Users\npard\Desktop\Idris_Examples> b
True
False
```

TIPOS Y FUNCIONES TIPOS DEPENDIENTES

En Idris, los tipos son de **primera clase**, lo que significa que pueden ser **calculados, manipulados y pasados a funciones como cualquier otra construcción del lenguaje**.

Por ejemplo, podemos escribir una función que calcula tipos:

```
isSingleton : Bool -> Type
isSingleton True = Nat
isSingleton False = List Nat

sum : (single : Bool) -> isSingleton single -> Nat
sum True x = x
sum False [] = 0
sum False (x :: xs) = x + sum False xs

main : IO ()
main = do
    putStrLn (show (sum True 5))
    putStrLn (show (sum False [5,6,7]))
```

```
PS C:\Users\npard\Desktop\Idris_Examples> build\exec\Hello_World
5
18
```

TIPOS Y FUNCIONES TIPOS DEPENDIENTES

```
module Main
import Data.String

divide : (n : Double) -> (m : Double) -> Maybe Double
divide n 0 = Nothing
divide n m = Just (n / m)

parseInput : String -> Maybe Double
parseInput str = case Data.String.parseDouble str of
    Just n => Just (fromDouble n)
    Nothing => Nothing
```

Funciones

```
main : IO ()
main = do putStrLn "Enter a number: "
          a <- getLine
          let num1 = case parseInput a of
              Just n => n
              Nothing => 0.0
          putStrLn "Enter another number: "
          b <- getLine
          let num2 = case parseInput b of
              Just n => n
              Nothing => 0.0
          putStrLn (show (divide num1 num2))
```

Función **Main**

TIPOS Y FUNCIONES INPUT/OUTPUT

TIPO IO:

Las operaciones son ejecutadas externamente por el sistema, en tiempo de ejecución.

Recibe un String

```
module Main

sayHello : String -> String
sayHello name = "Hello " ++ name

sayBye : String -> String
sayBye name = " and bye " ++ name

main : IO ()
main = do putStrLn "What is your name? "
          name <- getLine
          putStrLn (sayHello name)
          putStrLn (sayBye name)
          putStrLn ("This is the end.")
```

```
putStrLn : String -> IO ()
putStr   : String -> IO ()

getLine : IO String
```

Imprime un String

Resultado

```
PS C:\Users\npard\Desktop\Idris_Exam
What is your name? Thomas
Hello Thomas and bye Thomas
This is the end.
PS C:\Users\npard\Desktop\Idris_Exam
```

INTERFACES

- Las interfaces son colecciones de funciones que predefinen un **comportamiento**.
- En idris permiten implementar funciones u operandos para **distintos tipos de datos**
- Idris permite la definición de implementaciones **por defecto** que permiten mayor flexibilidad al momento de su implementación.

SHOW
a → String

NAT
S (S Z)

"2"

BOOL
True

"True"

VECT
[1,2]

"[1,2]"

INTERFACES IMPLEMENTACIÓN

Definimos una
interfaz

```
interface Eq a where
    (==) : a -> a -> Bool
    (/=) : a -> a -> Bool
    x /= y = not (x == y)
    x == y = not (x /= y)
```

Definiciones
por defecto

Implementamos **Eq**
para el tipo **Nat**

```
Eq Nat where
    Z      == Z      = True
    (S x) == (S y) = x == y
    Z      == (S y) = False
    (S x) == Z      = False
    x /= y = not (x == y)
```

Implementamos **solo una** de
las definiciones por defecto

INTERFACES

EXTENSION

Se declara como
una interfaz que
extiende de **Eq**

```
data Ordering = LT | EQ | GT
interface Eq a => Ord a where
    compare : a -> a -> Ordering

    (<) : a -> a -> Bool
    (>) : a -> a -> Bool
    (≤) : a -> a -> Bool
    (≥) : a -> a -> Bool
    max : a -> a -> a
    min : a -> a -> a
```

VARIOS PARÁMETROS

INTERFACES IMPLEMENTACIÓN

Functor List where

```
map f []      = []
map f (x::xs) = f x :: map f xs
```

```
interface Functor (θ f : Type -> Type) where
  map : (m : a -> b) -> f a -> f b
```

Función
parámetro

Datos a
aplicarle **m**

INTERFACES APLICATIVOS Y MÓNADAS

```
infixl 2 <*>
```

```
interface Functor f => Applicative (0 f : Type -> Type) where
  pure   : a -> f a
  (<*>)  : f (a -> b) -> f a -> f b
```

```
interface Applicative m => Monad (m : Type -> Type) where
  (=>)  : m a -> (a -> m b) -> m b
```

INTERFACES APLICATIVOS Y MÓNADAS

```
ome > vboxuser > Desktop > Idris2 > ≡ hello.idr
1  f : List (Int -> Int)
2  f = [(+1), (*2), (`div` 3)]
3
4  x : List Int
5  x = [1, 2, 3]
6
7  result : List Int
8  result = f <*> x
```

Aplicativo

Monad Maybe where

`Nothing >>= k = Nothing`
`(Just x) >>= k = k x`

Mónada

MÓDULOS

PROGRAMA DE IDRIS

module A

name
imports

declaraciones /
definiciones
tipos interfaces
funciones

module B

name
imports

declaraciones /
definiciones
tipos interfaces
funciones

...

module Z

name
imports

declaraciones /
definiciones
tipos interfaces
funciones

EJEMPLO

```
module BTTree
public export
data BTTree a = Leaf
              | Node (BTTree a) a (BTTree a)

export
insert : Ord a => a -> BTTree a -> BTTree a
insert x Leaf = Node Leaf x Leaf
insert x (Node l v r) = if (x < v) then (Node (insert x l) v r)
                           else (Node l v (insert x r))

export
toList : BTTree a -> List a
toList Leaf = []
toList (Node l v r) = BTTree.toList l ++ (v :: BTTree.toList r)

export
toTree : Ord a => List a -> BTTree a
toTree [] = Leaf
toTree (x :: xs) = insert x (toTree xs)

module Main

import BTTree

main : IO ()
main = do let t = toTree [1,8,2,7,9,3]
          print (BTTree.toList t)
```

MÓDULOS NOMBRES

```

module BTTree
public export
data BTTree a = Leaf
  | Node (BTTree a) a (BTTree a)

export
insert : Ord a => a -> BTTree a -> BTTree a
insert x Leaf = Node Leaf x Leaf
insert x (Node l v r) = if (x < v) then (Node (insert x l) v r)
                           else (Node l v (insert x r))

export
toList : BTTree a -> List a
toList Leaf = []
toList (Node l v r) = BTTree.toList l ++ (v :: BTTree.toList r)

export
toTree : Ord a => List a -> BTTree a
toTree [] = Leaf
toTree (x :: xs) = insert x (toTree xs)

```

Para eliminar la ambigüedad de los nombres (en caso de tener los mismos nombres para distintos módulos) se pueden verificar los tipos o:

Calificar los nombres con el nombre del módulo:

- BTTree.**BTTree**
- BTTree.**Leaf**
- BTTree.**Node**
- BTTree.**insert**
- BTTree.**toList**
- BTTree.**toTree**

Usar la palabra reservada **with**

- with BTTree.**insert**
(insert x empty)
- with
[BTTree.**insert**,
BTTree.**empty**]
(insert x empty)

MÓDULOS EXPORTAR MODIFIERS

PRIVATE
(default)

EXPORT

PUBLIC EXPORT

Solo el **top level** es
exportado

La **definición** en
general es exportada

Funciones

Data Types

Interfaces

No se exporta

Se exporta **solo el tipo**

Se exporta **solo el
constructor del tipo**

Se exporta **solo el
nombre de la interfaz**

Se exporta **tipo y definición**

Se exporta el **constructor** del tipo
y de los **datos** dentro del tipo

Se exporta el **nombre, nombres
de métodos y definiciones**

NAMESPACES EXPLICITOS

```
module Foo

namespace X
  export
    test : Int -> Int
    test x = x * 2

namespace Y
  export
    test : String -> String
    test x = x ++ x
```

*Foo> test 3
6 : Int

Foo.X.test

*Foo> test "foo"
"foofoo" : String

Foo.Y.test

¿Es posible
sobrecargar
nombres en un
módulo?

PARAMETERISED BLOCKS

Los grupos de funciones pueden ser parametrizadas con cierto número de argumentos usando la declaración **parameters**.

```
parameters (x : Nat, y : Nat)
  addAll : Nat -> Nat
  addAll z = x + y + z
```

Así, **parameters** añade los parámetros declarados a cada función, tipo y/o constructor dentro del bloque.

desde **REPL** *

firma de tipo

```
*params> :t addAll
addAll : Nat -> Nat -> Nat -> Nat
```

definición

```
addAll : (x : Nat) -> (y : Nat) -> (z : Nat) -> Nat
addAll x y z = x + y + z
```

* Si se llama por fuera del bloque, los parámetros de la función se deben llamar **explícitamente**.

```
home > vboxuser > Desktop > Idris2 > ⊚ hello.idr
1  f : List (Int -> Int)
2  f = [ (+1), (*2), (`div` 3) ]
3
4  x : List Int
5  x = [1, 2, 3]
6
7  result : List Int
8  result = f <*> x
```

MUCHAS GRACIAS
POR SU ATENCIÓN

REFERENCIAS



1. **“Idris (lenguaje de programación),”** hmн.wiki, 2023.
[`https://hmн.wiki/es/Idris_\(programming_language\)`](https://hmн.wiki/es/Idris_(programming_language)) (accessed May 15, 2023).
2. **“Types and Functions – Idris 1.3.3 documentation,”** Idris-lang.org, 2023. [`https://docs.idris-lang.org/en/latest/tutorial/typesfuns.html`](https://docs.idris-lang.org/en/latest/tutorial/typesfuns.html) (accessed May 15, 2023).
3. **“Documentation for the Idris 2 Language – Idris2 0.0 documentation,”** Readthedocs.io, 2023. [`https://idris2.readthedocs.io/en/latest/index.html`](https://idris2.readthedocs.io/en/latest/index.html) (accessed May 15, 2023).