

El conjunto de los enteros en Haskell

Diferencia entre `Int` e `Integer` en Haskell

- `Int` es el tipo (conjunto de pertenencia) de los enteros que la computadora puede representar, esto depende de la arquitectura interna de cada máquina.
- `Integer` representa al conjunto de los enteros matemáticos.

El significado del operador `::` en Haskell es "tiene tipo". Entonces en Haskell, la expresión `8 :: Int` indica que 8 "tiene tipo" `Int`, es decir, el conjunto de pertenencia es `Int`, por lo tanto es un entero. Escriba `8 :: Integer` en Hugs y explique el resultado.

El beneficio de usar `Integer` es teórico, propiedades que se verifican en `Integer`, podrán no verificarse en `Int`.

Realizar las siguientes pruebas en Hugs:

1. Calcular 2^2
2. Calcular 2^4
3. Calcular 2^{32}
4. Calcular 2^{64}
5. Calcular 2^{200}
6. Calcular $2^4 :: \text{Int}$
7. Calcular $2^{32} :: \text{Int}$
8. Calcular $2^{31} :: \text{Int}$
9. Calcular $2^{31} - 1 :: \text{Int}$

Estudiar los resultados obtenidos y analizar posibles explicaciones

Abrir en el bloc de notas el archivo `primero.hs`

Copiar al archivo `primero.hs` las siguientes definiciones de funciones:

```
f :: Int -> Int
f n = 2^n
```

```
g :: Int -> Integer
g n = 2^n
```

Estudiar su significado.

1. Calcular $f\ 2$
2. Calcular $g\ 4$
3. Calcular $f\ 32$
4. Calcular $f\ 64$
5. Calcular $f\ 200$
6. Calcular $g\ 200$
7. Calcular $g\ 35$
8. Otras aplicaciones de f o g que deseen

Explicar los resultados.

Haskell usa el símbolo `=` para indicar "se define como" y `==` para la igualdad. El símbolo `/=` indica "distinto de".

Ejemplo:

`divide d n = mod n d == 0` (`mod` es la función predefinida en Haskell que devuelve el resto de la división entre dos números)

La línea anterior define la función "divide", `d` y `n` se denominan sus argumentos o parámetros.

El conjunto Bool en Haskell

Hugs evalúa expresiones booleanas:

Escriba y ejecute las siguientes expresiones en Hugs.

- 1) `7<0`
- 2) `8>2`
- 3) `div 8 4 == 2`
- 4) `div 4 3 == 0`
- 5) `mod 8 4 == 0`
- 6) `not (mod 8 4 == 7)`
- 7) `4 /= 4`
- 8) `div 8 9 /= 20`
- 9) `mod 50 10 <= 0`
- 10) `div 50 10 >= 6`

Más expresiones booleanas:

`&&` - and lógico

`&&`: `BoolxBool` → `Bool`

Notación currificada:

`&&`: `Bool` → `Bool` → `Bool`

`||` - or lógico

`||`: `BoolxBool` → `Bool`

Notación currificada:

`||`: `Bool` → `Bool` → `Bool`

Los operadores and y or utilizan notación infija

`not` - not lógico

`not`: `Bool` → `Bool`

El operador not utiliza notación prefija

Calcular en Haskell:

- 1) `(8>2) || (4>1)`
- 2) `(8>2) && (4>1)`
- 3) `(8>2) && (4<1)`
- 4) Escribir y probar en Hugs sus propias expresiones utilizando los operadores lógicos introducidos.

Los conjuntos char y string en Haskell

El tipo de los caracteres se denomina char.

Al tipo de las tiras de caracteres se les denomina string.

El comando de Hugs `:type <expr>` nos devuelve el tipo de una expresión, es útil para saber si el tipo de una expresión que queremos usar se corresponde con lo que pensamos.

Calcular en Haskell:

- 1) `:type 'a'`
- 2) `:type "abcdf"`
- 5) `:type True`
- 3) `:type 4<5`
- 4) `:type "a"`
- 5) `length "perro"`
- 6) Indicar tipo de `length` e indicar qué toma y que devuelve.
- 7) `:type length`
- 8) Escribir y probar en Hugs sus propias expresiones.

Estructuras de datos básicas: tuplas

Una de las estructuras de datos más usadas en Haskell son las tuplas.

Tuplas

Las tuplas se escriben como una secuencia de valores de datos, separados por coma, entre paréntesis.

Ejemplo (4, 'a', "gato")

Probar en Hugs:

- 1) (4, 'a', "gato")
- 2) :type (4, 'a', "gato")
- 3) :type ('a', "zapato")
- 4) :type ("haskell", "matematica")
- 5) :type (True, 5)
- 6) :type ('?', "pepito")
- 7) :type ("cualquiera", (3.14, False))

En general, si $x::A$ e $y::B$, entonces $(x, y)::(A, B)$

Probar en hugs:

- 1) fst (6, 'b')
- 2) snd (6, 'b')

Explicar las funciones fst y snd, dar tipo (qué recibe y qué devuelve)

CONTINUARÁ...