

## Módulos

Cuando creamos programas, es una buena práctica que las funciones y tipos que de alguna forma están relacionados estén en el mismo módulo. De esta forma, podemos fácilmente reutilizar esas funciones en otros programas importando esos módulos.

### Ejemplo 1:

Escribir el siguiente código en modulo.hs:

```
module Funciones where

esPar :: Int -> Bool
esPar n | (mod n 2 == 0) = True
        | otherwise = False

esImPar :: Int -> Bool
esImPar n | (mod n 2 == 0) = False
           | otherwise = True
```

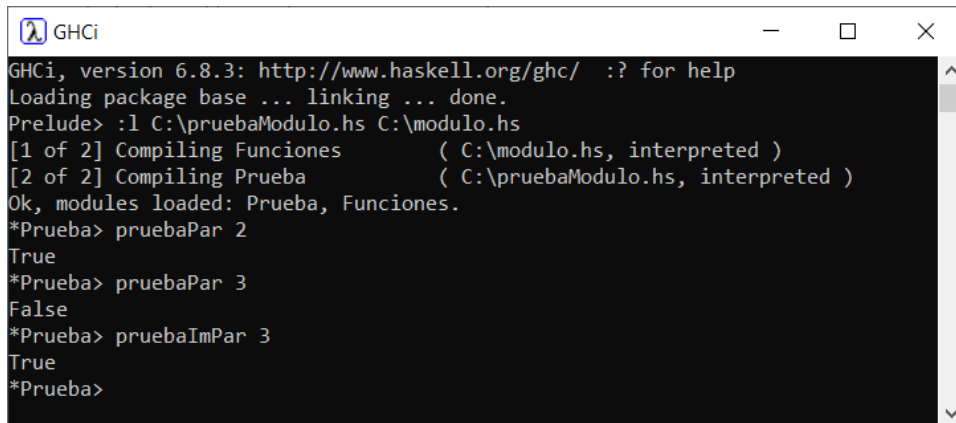
Escribir el siguiente código en pruebaModulo.hs

```
module Prueba where

import Funciones

pruebaPar n = esPar n
pruebaImPar n = esImPar n
```

Probar:



```

GHCi
GHCi, version 6.8.3: http://www.haskell.org/ghc/  :? for help
Loading package base ... linking ... done.
Prelude> :l C:\pruebaModulo.hs C:\modulo.hs
[1 of 2] Compiling Funciones      ( C:\modulo.hs, interpreted )
[2 of 2] Compiling Prueba        ( C:\pruebaModulo.hs, interpreted )
Ok, modules loaded: Prueba, Funciones.
*Prueba> pruebaPar 2
True
*Prueba> pruebaPar 3
False
*Prueba> pruebaImPar 3
True
*Prueba>
```

A través del módulo pruebaModulo.hs estamos utilizando funciones implementadas en modulo.hs

Si no se quiere que otros módulos utilicen la función esImPar:

```
module Funciones (esPar) where

esPar :: Int -> Bool
esPar n | (mod n 2 == 0) = True
        | otherwise = False

esImPar :: Int -> Bool
esImPar n | (mod n 2 == 0) = False
          | otherwise = True
```

Para especificar ambas funciones:

```
module Funciones (esPar,esImPar) where

esPar :: Int -> Bool
esPar n | (mod n 2 == 0) = True
        | otherwise = False

esImPar :: Int -> Bool
esImPar n | (mod n 2 == 0) = False
          | otherwise = True
```

Módulos externos solamente acceden a lo que está entre paréntesis

Nota: También se puede añadir lista de importación tras el nombre del módulo

```
import Funciones (esPar)
```

Se puede ocultar con:

```
import Funciones hiding (esPar)
```

**Ejemplo 2:** Módulo que exporte funciones que nos permitan calcular el volumen y el área de algunos objetos geométricos. Empezaremos creando un fichero llamado `Geometry.hs`.

Especificamos el nombre de un módulo al principio del módulo. Si hemos llamado al fichero `Geometry.hs` debemos darle el nombre de `Geometry` a nuestro módulo. Luego, especificamos la funciones que se exportan, y luego comenzamos a definir dichas funciones.

Vamos a calcular el área y el volumen de las esferas, cubos y hexaedros. Definamos estas funciones:

```

module Geometry
( sphereVolume
, sphereArea
, cubeVolume
, cubeArea
, cuboidArea
, cuboidVolume
) where

sphereVolume :: Float -> Float
sphereVolume radius = (4.0 / 3.0) * pi * (radius ^ 3)

sphereArea :: Float -> Float
sphereArea radius = 4 * pi * (radius ^ 2)

cubeVolume :: Float -> Float
cubeVolume side = cuboidVolume side side side

cubeArea :: Float -> Float
cubeArea side = cuboidArea side side side

cuboidVolume :: Float -> Float -> Float -> Float
cuboidVolume a b c = rectangleArea a b * c

cuboidArea :: Float -> Float -> Float -> Float
cuboidArea a b c = rectangleArea a b * 2 + rectangleArea a c * 2 + rectangleArea c b * 2

rectangleArea :: Float -> Float -> Float
rectangleArea a b = a * b

```

Como un cubo es un caso especial de un hexaedro, hemos definido el área y el volumen tratándolo como un hexaedro con todos sus lados iguales. También hemos definido una función auxiliar llamada `rectangleArea`, la cual calcula el área de un rectángulo basándose en el tamaño de sus lados. Hemos utilizado esta función en las funciones `cuboidArea` y `cuboidVolume` pero no la hemos exportado. Esto es debido a que queremos que nuestro módulo solo muestre funciones para tratar estos tres objetos dimensionales, hemos utilizado `rectangleArea` pero no la hemos exportado.

Cuando estamos creando un módulo, normalmente exportamos solo las funciones que actúan como una especie de interfaz de nuestro módulo de forma que la implementación se mantenga oculta. Si alguien usa nuestro módulo `Geometry`, no nos tenemos que preocupar por funciones las funciones que no exportamos. Podemos decidir cambiar esas funciones por completo o eliminarlas a cambio de una nueva versión (podríamos eliminar `rectangleArea` y utilizar `*`) y nadie se daría cuenta ya que no las estamos exportando.

Para utilizar nuestro módulos simplemente usamos:

```
import Geometry
```

Aunque `Geometry.hs` debe estar en el mismo directorio que el programa que lo está utilizando.

También podemos dar a los módulos una estructura jerárquica. Cada módulo puede tener cualquier número de submódulos y ellos mismo pueden tener cualquier otro número de submódulos. Vamos a dividir las funciones del módulo `Geometry` en tres submódulos de forma de cada objeto tenga su propio módulo.

Primero creamos un directorio llamado Geometry. Mantén la G en mayúsculas. Dentro de él crearemos los ficheros sphere.hs, cuboid.hs, y cube.hs. Este será el contenido de los ficheros:

sphere.hs

```
module Geometry.Sphere
( volume
, area
) where

volume :: Float -> Float
volume radius = (4.0 / 3.0) * pi * (radius ^ 3)

area :: Float -> Float
area radius = 4 * pi * (radius ^ 2)
```

cuboid.hs

```
module Geometry.Cuboid
( volume
, area
) where

volume :: Float -> Float -> Float -> Float
volume a b c = rectangleArea a b * c

area :: Float -> Float -> Float -> Float
area a b c = rectangleArea a b * 2 + rectangleArea a c * 2 + rectangleArea c b * 2

rectangleArea :: Float -> Float -> Float
rectangleArea a b = a * b
```

cube.hs

```
module Geometry.Cube
( volume
, area
) where

import qualified Geometry.Cuboid as Cuboid

volume :: Float -> Float
volume side = Cuboid.volume side side side

area :: Float -> Float
area side = Cuboid.area side side side
```

Fíjate que primero hemos creado una carpeta llamada Geometry y luego y luego hemos definido el nombre como Geometry.Sphere. Hicimos lo mismo con el hexaedro. Fíjate también que en los tres módulos hemos definido funciones con los mismos nombres. Podemos hacer esto porque están separados en módulos distintos. Queremos utilizar las funciones de Geometry.Cuboid en Geometry.Cube pero no podemos usar simplemente `import Geometry.Cuboid` ya que importaríamos funciones con el mismo nombre que en Geometry.Cube. Por este motivo lo cualificamos.

Así que si ahora estamos en un fichero que se encuentra en el mismo lugar que la carpeta Geometry podemos utilizar:

```
import Geometry.Sphere
```

Y luego podemos utilizar `area` y `volume` y nos darán el área y el volumen de una esfera. Si queremos usar dos o más módulos de éstos, tenemos que cualificarlos para que no haya conflictos con los nombres. Podemos usar algo como:

```
import qualified Geometry.Sphere as Sphere
import qualified Geometry.Cuboid as Cuboid
import qualified Geometry.Cube as Cube
```

Ahora podemos llamar a `Sphere.area`, `Sphere.volume`, `Cuboid.area`, etc. y cada uno calculará el área o el volumen de su respectivo objeto.