

Polinomios I

Taller de Álgebra I

Segundo cuatrimestre 2019

type

Definamos un renombre de tipos para Polinomios en $\mathbb{R}[X]$:

```
type Polinomio = [Float]
```

donde el elemento i de la lista (empezando desde la cola, contando desde cero) corresponderá con el coeficiente a_i del polinomio $P(X) = \sum_{i=0}^n a_i X^i$.
Por ejemplo el polinomio $X^2 + 3X - 1$ se representa por $[1, 3, (-1)]$.

- ▶ Invariante del tipo: La lista de coeficientes no tiene ceros a izquierda
 - ▶ Se puede suponer que este invariante se cumple con los parámetros que recibimos.
 - ▶ Se debe garantizar este invariante al construir polinomios.
- ▶ Implementar una función para calcular el grado de un polinomio.
`grado :: Polinomio -> Integer` (en general se usa la convención $\text{grado}(0) = -\infty$, en la implementación se puede dejar `grado []` indefinido).

```
Ejemplo> grado [3, 0, 0, 0, 1]  
4
```

```
Ejemplo> grado []  
*** Exception: Prelude.undefined
```

Evaluación y derivada de polinomios

Ejercicios

- 1 Implementar una función

`evaluar :: Polinomio -> Float -> Float` que dado $P \in \mathbb{R}[X]$ y $X \in \mathbb{R}$ calcule $P(X)$

```
Ejemplo> evaluar [1, 0, 0, 0] 2  
8.0
```

```
Ejemplo> evaluar [4, 3, 2] 5  
117.0
```

- 2 Implementar una función para calcular el polinomio derivado.

`derivada :: Polinomio -> Polinomio`

```
Ejemplo> derivada [1, 5, 3]  
[2.0,5.0]
```

- 3 Implementar una función para calcular el polinomio derivado N veces.

`derivadaN :: Integer -> Polinomio -> Polinomio`

```
Ejemplo> derivadaNesima 2 [11, 5, 3]  
[22.0]
```

Suma y producto por escalar

Ejercicios

- 1 Implementar una función que sume dos polinomios.

```
suma :: Polinomio -> Polinomio -> Polinomio
```

```
Ejemplo> suma [1, 5, 3] [2, 3, (-1), 0]  
[2.0,4.0,4.0,3.0]
```

```
Ejemplo> suma [2, 1, 1] [(-2), 3, 4]  
[4.0,5.0]
```

Observar que se deben eliminar los 0s a la izquierda. Sugerencia:

```
suma f g = limpiar (sumaAux f g)
```

- 2 Implementar una función para calcular el producto de un escalar por un polinomio.

```
productoPorEscalar :: Float -> Polinomio -> Polinomio
```

```
Ejemplo> productoPorEscalar 5 [2, 3, (-1)]  
[10.0,15.0,-5.0]
```

Ejercicios

- 1 Implementar una función para calcular el producto de un monomio aX^n (con $a \in \mathbb{R}, n \in \mathbb{N}$) por un polinomio.

```
productoPorMonomio :: (Float, Integer) -> Polinomio -> Polinomio
```

```
Ejemplo> productoPorMonomio (2.5, 2) [2, 4, (-2)]  
[5.0,10.0,-5.0,0.0,0.0]
```

- 2 Implementar una función para calcular el producto de dos polinomios.

```
producto :: Polinomio -> Polinomio -> Polinomio
```

```
Ejemplo> producto [2, 3, (-1)] [1, (-3)]  
[2.0,-3.0,-10.0,3.0]
```

Complejos

type

Definamos un renombre de tipos para Complejos

```
type Complejo = (Float, Float)
```

donde el primer elemento es la parte real y el segundo la parte imaginaria.

Ejercicio

Implementar las funciones

- 1 suma :: Complejo -> Complejo -> Complejo
- 2 producto :: Complejo -> Complejo -> Complejo
- 3 potencia :: Complejo -> Integer -> Complejo

```
Ejemplo> suma (2,3) (-1,4)  
(1.0,7.0)
```

```
Ejemplo> producto (2,3) (-1,4)  
(-14.0,5.0)
```

Ejercicio

Implementar una función

1 `evaluarComplejo :: Polinomio -> Complejo -> Complejo`
que evalúe un polinomio (con coeficientes reales) en un número complejo.

```
Ejemplo> evaluarComplejo [1,0,1] (0,(-1))  
(0,0)
```