



Los siguientes ejercicios constituyen el Taller 1. Para aprobar el taller, deben resolver todos los ejercicios. Un ejercicio se considera resuelto si pasa todos los tests que les proveemos.

Recuerden: No pueden utilizar los métodos de “String” que no sean la concatenación (`s ++ s`), el largo (`length()`) y el acceso al *i*-ésimo elemento (`charAt(indice)`). Además de las operaciones básicas de Java, pueden utilizar la biblioteca “Math”. Les recomendamos buscar cómo utilizar los métodos de esta biblioteca.

Link de entrega: Campus Virtual (subir **únicamente** el archivo `Funciones.java`)

Última fecha de entrega: domingo 31/08

1. Primera parte: Funciones en java

Ejercicio 1. Implementar la función `cuadrado`, que devuelve el resultado de multiplicar a un número por si mismo

```
proc cuadrado (in x:  $\mathbb{Z}$ ) :  $\mathbb{Z}$  {  
    requiere {true}  
    asegura {res =  $x * x$ }  
}
```

Ejercicio 2. Implementar la función `distancia`, que dadas las coordenadas $(x, y) \in \mathbb{R}^2$, devuelve la distancia al origen de coordenadas.

```
proc distancia (in x:  $\mathbb{R}$ , in y:  $\mathbb{R}$ ) :  $\mathbb{R}$  {  
    requiere {true}  
    asegura {res =  $\sqrt{x * x + y * y}$ }  
}
```

Ejercicio 3. Implementar la función `esPar`, que determina si un número natural es par.

```
proc esPar (in n:  $\mathbb{N}$ ) : Bool {  
    requiere {true}  
    asegura {res = true  $\leftrightarrow$  divideA(2, n)}  
    pred divideA (d:  $\mathbb{N}$ , n:  $\mathbb{N}$ ) {  
        ( $\exists k : \mathbb{N}$ ) ( $n = d * k$ )  
    }  
}
```

Si hicieron este ejercicio usando `if`, `for` o `while`, piensen cómo hacerlo sin usarlo.

Ejercicio 4. Implementar la función `esBisiesto`, que determina si un año es bisiesto. Los años bisiestos son los múltiplos de 4 que no son múltiplos de 100. Esta regla tiene una excepción: los años múltiplo de 400 se consideran bisiestos de todos modos.

```
proc esBisiesto (in n:  $\mathbb{N}$ ) : Bool {  
    requiere {true}  
    asegura {res = true  $\leftrightarrow$  ( $\text{divideA}(4, n) \wedge \neg \text{divideA}(100, n)$ )  $\vee$   $\text{divideA}(400, n)$ }  
}
```

Si hicieron este ejercicio usando `if`, `for` o `while`, piensen cómo hacerlo sin usarlo.

Ejercicio 5. Implementar la función `factorial`, que calcula el factorial de un número natural. Recordar que $0! = 1$ por definición.

```
proc factorial (in n:  $\mathbb{N}$ ) :  $\mathbb{N}$  {  
    requiere {true}  
    asegura {res =  $\prod_{i=1}^n i$ }  
}
```

Pensar tanto una solución recursiva como una iterativa.

Ejercicio 6. Implementar la función `esPrimo`, que determina si un número natural es primo. Recordar que un número es primo cuando tiene exactamente dos divisores.

```
proc esPrimo (in n: ℕ) : Bool {
  requiere {true}
  asegura {res = true ↔ primo(n)}
  pred primo (n: ℕ) {
     $\left(\sum_{i=1}^n \text{if } \text{divideA}(i, n) \text{ then } 1 \text{ else } 0 \text{ fi}\right) = 2$ 
  }
}
```

Ejercicio 7. Implementar la función `sumatoria`, que dado un arreglo de números enteros, calcula la suma de sus elementos.

```
proc sumatoria (in l: seq(ℤ)) : ℤ {
  requiere {true}
  asegura {res =  $\sum_{i=0}^{|l|-1} l[i]$ }
}
```

Ejercicio 8. Implementar la función `busqueda`, que dado un arreglo de números enteros l y un número *buscado*, devuelve alguna de las posiciones del arreglo en las que se encuentra el número *buscado*.

```
proc busqueda (in l: seq(ℤ), in buscado: ℤ) : ℕ {
  requiere {(∃i : ℕ) (i < |l| ∧ l[i] = buscado)}
  asegura {l[res] = buscado}
}
```

Ejercicio 9. Implementar la función `tienePrimo`, que determina si un arreglo de números naturales contiene un número primo.

```
proc tienePrimo (in l: seq(ℕ)) : Bool {
  requiere {true}
  asegura {(∃i : ℕ) (i < |l| ∧ primo(l[i]))}
}
```

Ejercicio 10. Implementar la función `todosPares`, que determina si todos los números de un arreglo de números son pares.

```
proc todosPares (in l: seq(ℤ)) : Bool {
  requiere {true}
  asegura {(∀i : ℕ) (i < |l| → divideA(2, l[i]))}
}
```

Ejercicio 11. Implementar la función `esPrefijo`, que dados dos strings determina si el primero es prefijo del segundo.

```
proc esPrefijo (in s1: String, in s2: String) : Bool {
  requiere {true}
  asegura {(∀i : ℕ) (i < |s1| → (i < |s2| ∧ s1[i] = s2[i]))}
}
```

Ejercicio 12. Implementar la función `esSufijo`, que dados dos strings determina si el primero es sufijo del segundo.

```
proc esSufijo (in s1: String, in s2: String) : Bool {
  requiere {true}
  asegura {(∀i : ℕ) (i < |s1| → (i < |s2| ∧ s1[|s1| - i - 1] = s2[|s2| - i - 1]))}
}
```

Si la solución no usa `esPrefijo`, pensar una solución que lo use.

2. Segunda parte: Debugging

Los siguientes ejercicios tienen ya un código que los implementa, aunque con algunos errores. Su objetivo es encontrar estos errores y arreglarlos para que pasen los tests.

Ejercicio 13. Debuggear el código que implementa el operador **xor** (que se simboliza con \vee). Este operador representa un “o exclusivo”: $A \vee B$ es verdadero cuando A ó B es verdadero, pero no ambos. Cumple la siguiente tabla de verdad:

A	B	$A \vee B$
false	false	false
false	true	true
true	false	true
true	true	false

Ejercicio 14. Debuggear el código que implementa la función **iguales**, que determina si dos arreglos de número enteros son iguales.

```
proc iguales (in xs: seq<Z>, in ys: seq<Z>) : Bool {  
  requiere {true}  
  asegura {res = true  $\leftrightarrow$  |xs| = |ys|  $\wedge_L$  ( $\forall i : \mathbb{N}$ ) ( $i < |xs| \rightarrow_L xs[i] = ys[i]$ )}  
}
```

Ejercicio 15. Debuggear el código que implementa la función **todosPositivos**, que determina si todos los valores de un arreglo de número enteros son positivos.

```
proc todosPositivos (in xs: seq<Z>) : Bool {  
  requiere {true}  
  asegura {res = true  $\leftrightarrow$  ( $\forall x : \mathbb{N}$ ) ( $x \in xs \rightarrow 0 < x$ )}  
}
```

Ejercicio 16. Debuggear el código que implementa la función **maximo**, que devuelve el máximo valor de un arreglo de números enteros.

```
proc maximo (in xs: seq<Z>) : Z {  
  requiere {|xs| > 0}  
  asegura {res  $\in xs \wedge$  ( $\forall x : \mathbb{N}$ ) ( $x \in xs \rightarrow x \leq res$ )}  
}
```

Ejercicio 17. Debuggear el código que implementa la función **ordenado**, que determina si un arreglo de número enteros está ordenado de manera creciente.

```
proc ordenado (in xs: seq<Z>) : Bool {  
  requiere {true}  
  asegura {res = true  $\leftrightarrow$  ( $\forall i, j : \mathbb{N}$ ) ( $i < j \wedge j < |xs| \rightarrow_L xs[i] \leq xs[j]$ )}  
}
```