Práctica 9: Programación funcional en Swift: clausuras y funciones de orden superior

Entrega de la práctica

Para entregar la práctica debes subir a Moodle el fichero `practica09.swift` con una cabecera inicial con tu nombre y apellidos, y las soluciones de cada ejercicio separadas por comentarios. Cada solución debe incluir:

- La **definición de las funciones** que resuelven el ejercicio.
- Una visualización por pantalla de todos los ejemplos incluidos en el enunciado que **demuestren qué hace la función**.

Ejercicios

> ⚠ **Importante**
>
> Antes de empezar la práctica debes estudiar los apartados de teoría del tema de Programación Funcional en Swift:
>
> - Opcionales [https://domingogallardo.github.io/apuntes-lpp/teoria/tema05-programacion-funcional-swift/tema05-programacion-funcional-swift.html#opcionales]
> - Clausuras [https://domingogallardo.github.io/apuntes-lpp/teoria/tema05-programacion-funcional-swift/tema05-programacion-funcional-swift.html#clausuras]
> - Funciones de orden superior [https://domingogallardo.github.io/apuntes-lpp/teoria/tema05-programacion-funcional-swift/tema05-programacion-funcional-swift.html#funciones-de-orden-superior]
> - Genéricos [https://domingogallardo.github.io/apuntes-lpp/teoria/tema05-programacion-funcional-swift/tema05-programacion-funcional-swift.html#genericos]

Ejercicio 1

a) Indica qué devuelven las siguientes expresiones:

a.1)

```
let nums = [1,2,3,4,5,6,7,8,9,10]
nums.filter{$0 % 3 == 0}.count
```

a.2)

```
let nums2 = [1,2,3,4,5,6,7,8,9,10]
nums2.map{$0+100}.filter{$0 % 5 == 0}.reduce(0,+)
```

a.3)

```
let cadenas = ["En", "un", "lugar", "de", "La", "Mancha"]
cadenas.sorted{$0.count < $1.count}.map{$0.count}
```

a.4)

```
let cadenas2 = ["En", "un", "lugar", "de", "La", "Mancha"]
cadenas2.reduce([]) {
    (res: [(String, Int)], c: String) -> [(String, Int)] in
        res + [(c, c.count)]}.sorted(by: {$0.1 < $1.1})
```

b) Explica qué hacen las siguientes funciones y pon un ejemplo de su funcionamiento:

b.1)

```
func f(nums: [Int], n: Int) -> Int {
    return nums.filter{$0 == n}.count
}
```

b.2)

```
func g(nums: [Int]) -> [Int] {
    return nums.reduce([], {
        (res: [Int], n: Int) -> [Int] in
            if !res.contains(n) {
```

```
                return res + [n]
            } else {
                return res
            }
    })
}
```

b.3)

```
func h(nums: [Int], n: Int) -> ([Int], [Int]) {
   return nums.reduce(([],[]), {
       (res: ([Int],[Int]), num: Int ) -> ([Int],[Int]) in
           if (num >= n) {
               return (res.0, res.1 + [num])
           } else {
               return ((res.0 + [num], res.1))
           }
   })
}
```

c) Implementa las siguientes funciones con funciones de orden superior.

c.1) Función `suma(palabras:contienen:)` :

```
suma(palabras: [String], contienen: Character) -> Int
```

que recibe una array de cadenas y devuelve la suma de las longitudes de las cadenas que contiene el carácter que se pasa como parámetro.

c.2) Función `sumaMenoresMayores(nums:pivote:)` :

```
sumaMenoresMayores(nums: [Int], pivote: Int) -> (Int, Int)
```

que recibe un array de números y un número pivote y devuelve una tupla con la suma de los números menores y mayores o iguales que el pivote.

Ejercicio 2

Define un tipo enumerado con un árbol genérico, tal y como hicimos en el último ejercicio de la práctica anterior, que tenga como genérico el tipo de dato que contiene.

En el siguiente ejemplo vemos cómo debería poderse definir con el mismo tipo genérico un árbol de enteros y un árbol de cadenas:

```
let arbolInt: Arbol = .nodo(53,
                            [.nodo(13, []),
                             .nodo(32, []),
                             .nodo(41,
                                   [.nodo(36, []),
                                    .nodo(39, [])
                                   ])
                            ])
let arbolString: Arbol = .nodo("Zamora",
                               [.nodo("Buendía",
                                      [.nodo("Albeza", []),
                                       .nodo("Berenguer", []),
                                       .nodo("Bolardo", [])
                                      ]),
                                .nodo("Galván", [])
                               ])
```

Define las funciones genéricas `toArray` y `toArrayFOS` que devuelvan un array con todos los componentes del árbol usando un recorrido *preorden* (primero la raíz y después los hijos). La primera la debes implementar con recursión mutua y la segunda usando funciones de orden superior.

Ejemplo:

```
print(toArray(arbol: arbolInt))
// Imprime: [53, 13, 32, 41, 36, 39]
print(toArrayFOS(arbol: arbolString))
// Imprime: ["Zamora", "Buendía", "Albeza", "Berenguer",
"Bolardo", "Galván"]
```

Ejercicio 3

Implementa en Swift la función `imprimirListadosNotas(alumnos:)` que recibe un array de tuplas, en donde cada tupla contiene información de la evaluación de un alumno de LPP (nombreAlumno, notaParcial1, notaParcial2, notaParcial3, añosMatriculacion) y que debe imprimir por pantalla los siguientes listados:

- listado 1: array ordenado por nombre del alumno (orden alfabético creciente)

- listado 2: array ordenado por la nota del parcial 1 (orden decreciente de nota)
- listado 3: array ordenado por la nota del parcial 2 (orden creciente de nota)
- listado 4: array ordenado por año de matriculación y nota del parcial 3 (orden decreciente de año y nota)
- listado 5: array ordenado por nota final (media de los tres parciales, ponderados en: 0,34, 0,33, 0,33) (orden decreciente de nota final)

Las ordenaciones hay que realizarlas usando la función `sorted`.

> **Nota**
>
> Para que los listados se muestren formateados con espacios, puedes usar la siguiente función (para ello también debes incluir el import que se indica)

```
import Foundation

func imprimirListadoAlumnos(_ alumnos: [(String, Double, Double, Double,
Int)]) {
    print("Alumno   Parcial1   Parcial2   Parcial3  Años")
    for alu in alumnos {
        alu.0.withCString {
            print(String(format:"%-10s %5.2f      %5.2f    %5.2f  %3d",
$0, alu.1,alu.2,alu.3,alu.4))
        }
    }
}
```

Ejemplo:

```
let listaAlumnos = [("Pepe", 8.45, 3.75, 6.05, 1),
                    ("Maria", 9.1, 7.5, 8.18, 1),
                    ("Jose", 8.0, 6.65, 7.96, 1),
                    ("Carmen", 6.25, 1.2, 5.41, 2),
                    ("Felipe", 5.65, 0.25, 3.16, 3),
                    ("Carla", 6.25, 1.25, 4.23, 2),
                    ("Luis", 6.75, 0.25, 4.63, 2),
                    ("Loli", 3.0, 1.25, 2.19, 3)]
imprimirListadosNotas(listaAlumnos)
```

Algunos de los listados que se deben mostrar serían los siguientes:

```

```

```
LISTADO ORIGINAL
Alumno   Parcial1   Parcial2   Parcial3  Años
Pepe        8.45       3.75       6.05    1
Maria       9.10       7.50       8.18    1
Jose        8.00       6.65       7.96    1
Carmen      6.25       1.20       5.41    2
Felipe      5.65       0.25       3.16    3
Carla       6.25       1.25       4.23    2
Luis        6.75       0.25       4.63    2
Loli        3.00       1.25       2.19    3

LISTADO ORDENADO por Parcial1 (decreciente)
Alumno   Parcial1   Parcial2   Parcial3  Años
Loli        3.00       1.25       2.19    3
Felipe      5.65       0.25       3.16    3
Carmen      6.25       1.20       5.41    2
Carla       6.25       1.25       4.23    2
Luis        6.75       0.25       4.63    2
Jose        8.00       6.65       7.96    1
Pepe        8.45       3.75       6.05    1
Maria       9.10       7.50       8.18    1
```

Ejercicio 4

Dado el array `listaAlumnos` del ejercicio anterior, utiliza funciones de orden superior para obtener los datos requeridos en cada caso.

A) Número de alumnos que han aprobado primer parcial y suspendido el segundo

```
print(listaAlumnos. _______________________________ )
// Resultado: 5
```

B) Alumnos que han aprobado la asignatura (tienen una nota final >= 5)

```
print(listaAlumnos._______________________________ )

// Resultado: ["Pepe", "Maria", "Jose"]
```

C) Nota media de todos los alumnos en forma de tupla `(media_p1, media_p2, media_p3)`

```

```

```
var tupla = listaAlumnos._____________________________________
)
tupla = (tupla.0 / Double(listaAlumnos.count), tupla.1 /
Double(listaAlumnos.count), tupla.2 /
Double(listaAlumnos.count))
print(tupla)
// Resultado: (6.6812499999999995, 2.7624999999999997,
5.2262500000000003)
```

Ejercicio 5

Implementa la función `construye` con el siguiente perfil:

```
func construye(operador: Character) -> (Int, Int) -> Int
```

La función recibe un operador que puede ser uno de los siguientes caracteres: `+`, `-`, `*`, `/` y debe devolver una clausura que reciba dos argumentos y realice la operación indicada sobre ellos.

Ejemplo:

```
var f = construye(operador: "+")
print(f(2,3))
// Imprime 5
f = construye(operador: "-")
print(f(2,3))
// Imprime -1
```
