

▼ Clase 10: Estructuras de Datos Recursivas II (Cap. 9)

▼ Listas que contienen estructuras

Definiremos un `registro` como una estructura compuesta de un campo de tipo texto para almacenar el nombre del producto, y de un campo de tipo numérico para almacenar el valor de dicho producto. Así pues, diseñemos la estructura:

```
# registro: producto(str) precio(int)
import estructura
estructura.crear("registro", "producto precio")
```

Más aún, podemos definir una `colección de registros` para almacenar toda la información que disponemos. A esto nos referiremos en este problema como **inventario**:

```
# inventario: [registro]*
# inventario es una lista de registros de largo indeterminado
```

```
# registro: producto(str) precio(int)
import estructura
from lista import *
estructura.crear("registro", "producto precio")
```

Es decir, un **inventario** está compuesto de:

1. Una lista vacía: `listaVacía`, o bien
2. Una lista que contiene un registro, encadenada al inventario:

```
crearLista(registro, inventario).
```

▼ Problema: Crear una función que sume todos los precios de un inventario

```
# suma: inventario -> int
# calcula la suma de todos los precios en unInventario
# suma(listaVacía) == 0
# suma(lista(registro("muneca", 2990), lista_vacia)) == 2990
# suma(lista(registro("robot", 5990), \
#   lista(registro("muneca", 2990), lista_vacia))) == 8980
def suma(unInventario):
```

```

def suma(unInventario):
    if vacia(unInventario):
        return 0
    else:
        item = cabeza(unInventario)
        return item.precio + suma(cola(unInventario))

# Tests
suma(listaVacia) == 0
suma(lista(registro("muneca", 2990), listaVacia)) == 2990
suma(lista(registro("robot", 5990), \
    lista(registro("muneca", 2990), listaVacia))) == 8980

```



True

▼ Funciones que producen listas

Recordemos la función `sueldo` que definimos anteriormente:

```

# sueldo: int -> int
# calcular el sueldo de un trabajador
# (a $4.500 la hora) que ha trabajado h horas
def sueldo(h):
    return 4500 * h

```

Esta función recibe como parámetro el número de horas trabajadas por un empleado, y produce su sueldo semanal. Por simplicidad, supondremos que todos los empleados ganan lo mismo por hora, es decir, \$4.500. Sin embargo, una empresa no está necesariamente interesada en una función como `sueldo`, que calcula el sueldo de un solo empleado, sino más bien en una función que calcule el sueldo total de todos sus empleados (sobre todo si hay muchos).

Llamemos a esta **función `listaSuealdos`**, tal que recibe una **lista de cuántas horas los empleados de la compañía han trabajado**, y devuelva una **lista de los sueldos semanales** por cada uno de ellos.

```

# listaSuealdos: lista(int) -> lista(int)
# crear una lista de sueldos semanales desde
# una lista de horas trabajadas (listaHoras)
def listaSuealdos(listaHoras):
    ...

```


Luego necesitamos ejemplos de entrada y salida:

```
listaSueldos(listaVacía)
```

```
listaSueldos(crearLista(10, listaVacía))
```

```
 lista(valor=45000, siguiente=None)
```

```
listaSueldos(crearLista(44, crearLista(10, listaVacía)))
```

```
 lista(valor=198000, siguiente=lista(valor=45000, siguiente=None))
```

▼Cuál sería la plantilla de diseño entonces?

```
# def listaSueldos(listaHoras):  
# if (vacía(listaHoras) == True):  
#     ...  
# else:  
#     ... cabeza(listaHoras)  
#     ... listaSueldos(cola(listaHoras)) ...
```

```
# listaSueldos: lista(int) -> lista(int)  
# crear una lista de sueldos semanales desde una  
# lista de horas trabajadas (listaHoras)  
def listaSueldos(listaHoras):  
    if vacía(listaHoras):  
        return listaVacía  
    else:  
        return crearLista(sueldo(cabeza(listaHoras)),\  
                           listaSueldos(cola(listaHoras)))
```

▼Otros ejemplos de funciones que devuelven una listas

```
from lista import *  
  
# listaX: int -> lista(int)  
def listaX(x):  
    if x > 10:  
        return listaVacía  
    else:
```

```
unaLista = lista(6, lista(8, listaVacía))
otraLista = lista(4, lista(5, listaVacía))
assert unionListas(unaLista, otraLista) == lista(6, lista(8, lista(4, lista(5, listaVacía))))
```


Escribimos la función:

```
# unionListas: lista(any) lista(any) -> lista(any)
# devuelve lista resultado de unir dos listas
# ejemplo: unionListas(lista(1, listaVacía), lista(2, listaVacía))
# devuelve lista(1, lista(2, listaVacía))
def unionListas(lista1, lista2):
    if vacía(lista1):
        return lista2
    else:
        return lista(cabeza(lista1), unionListas(cola(lista1), lista2))

# Test
unaLista = lista(1, listaVacía)
otraLista = lista(2, lista(3, listaVacía))
assert unionListas(unaLista, otraLista) == lista(1, lista(2, lista(3, listaVacía)))
```

```
return lista(x, listaX(x+1))
```


```
print (listaX(3)) # Que muestra en pantalla?
```

 lista(valor=3, siguiente=lista(valor=4, siguiente=lista(valor=5, siguiente=lista(valor=

```
# listaXY: int int -> lista(int)
def listaXY(x, y):
    if x > y:
        return listaVacia
    else:
        return lista(x, listaXY(x+1, y))
```

```
print (listaXY(1, 10)) # Que muestra en pantalla?
```

```
print (listaXY(7, 12)) # Que muestra en pantalla?
```

 lista(valor=1, siguiente=lista(valor=2, siguiente=lista(valor=3, siguiente=lista(valor=
lista(valor=7, siguiente=lista(valor=8, siguiente=lista(valor=9, siguiente=lista(valor=

▼ Funcion que copia una lista

Es decir crear otra lista que contenga los mismos valores:

```
# copiarLista: lista(any) -> lista(any)
# devuelve copia de lista
# ejemplo: copiarLista(lista(1, lista(2, listaVacia))) devuelve
# lista(1, lista(2, listaVacia))
def copiarLista(unalista):
    if vacia(unalista):
        return listaVacia
    else:
        return lista(cabeza(unalista), copiarLista(cola(unalista)))

# Test
assert copiarLista(lista(1, lista(2, listaVacia))) == lista(1, lista(2, listaVacia))
```

▼ Función para unir dos listas

Pensemos en los tests primero:

```
unaLista = lista(1, listaVacia)
otraLista = lista(2, lista(3, listaVacia))
assert unionListas(unaLista, otraLista) == lista(1, lista(2, lista(3, listaVacia)))
```