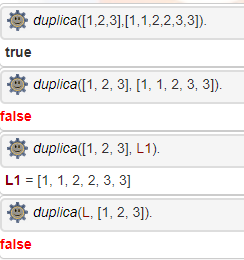
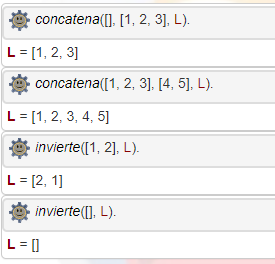
***Inteligencia Artificial: Práctica 3***

***Ejercicio 1:***

******

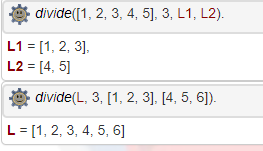
***Ejercicio 2:***



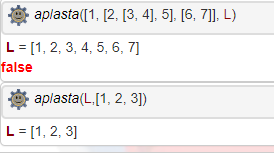
***Ejercicio 3:***



***Ejercicio 4:***



***Ejercicio 5:***



***Ejercicio 6:***

En este ejercicio se nos pide implementar una función **primos(N, L)**, en la que N es un número natural, y L es una lista que contiene sus divisores. Para implementarla hemos creado tres funciones:

1. NEXT\_FACTOR(N, F, NF) : esta función recibe el natural N, y dado un factor divisor F, calcula el siguiente NF, siendo la primera iteración con un 2 y la siguiente con 3, 5… y los números impares.



1. PRIMOS(N, L) : es la función principal, simplemente comprueba que N es mayor que 0 y llama a la función divisores, que es la encargada de la recursión, comenzando por probar el primo más pequeño, el 2.



1. DIVISORES(N, L, F): es la función recursiva en la que N es el factor a dividir, L contiene los divisores y F es el factor que probamos como divisor o no. Lo hemos dividido en 3 casos:

* *Caso base :* cuando N es 1 hemos llegado al final, pues no podemos seguir dividiendo el 1 en más primos.

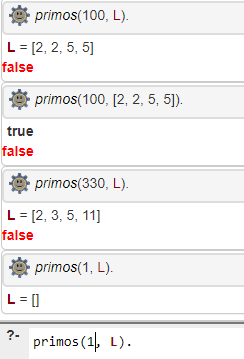


* *Caso 1:* F divide a N, es decir, N es 0 en módulo F. En este caso, dividimos F por F, metemos F en la lista de divisores y llamamos recursivamente a divisores con N/F y **sin cambiar el factor F**, pues puede tener multiplicidad mayor que 1.



* *Caso 2:* en este caso F no divide a N, por lo que no cambiamos N ni metemos F en la lista de los divisores, solamente calculamos el siguiente factor con next\_factor y llamamos a divisores con este nuevo factor NF.





***Ejercicio 7:***

En este ejercicio se nos pedía realizar 3 funciones:

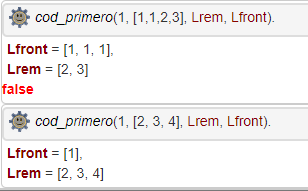
1. **COD\_PRIMERO(X, L, Lrem, Lfront)** : esta función consiste en dividir L en dos listas, en Lfront guardamos X y los elementos que sean iguales que X, mientras que en Lrem guardamos el resto de L.

Lo hemos implementado de la siguiente forma:

* *Caso base* : L y Lrem están vacías y Lfront solamente contiene X

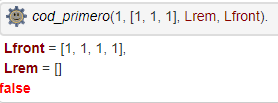


* *Caso 1* : el elemento coincide con el primer elemento de L, de modo que lo metemos en Lfront y llamamos recursivamente a la función con el resto de L, al que hemos llamado F.
* *Caso 2* : el elemento X no coincide con el primer elemento de L, esto quiere decir que ya hemos copiado en Lfront todos los elementos de L que eran iguales que X, de modo que solamente queda el resto de L, es decir, solo queda Lrem.

Hemos probado los siguientes ejemplos:

En este comprobamos que, si X está en L, guarda tanto el elemento como los repetidos en Lfront.

Éste comprueba que Lfront guarda el elemento, aunque no esté presente en la lista.



Por último, este ejemplo muestra que si L solamente contiene un único elemento repetido, Lrem sale vacía.

1. **COD\_ALL(L, L1) :** este apartado pedía implementar una función que utilizase cod\_primero, de tal modo que L (una lista formada por elementos repetidos o no, de modo que los repetidos aparecen juntos), es dividida en sublistas formadas por los elementos repetidos. L1 es una lista que contiene esas sublistas.

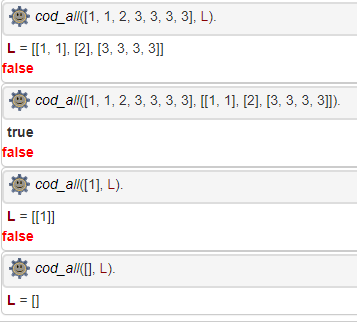
* *Caso base :* L1 y L están vacías, por lo que no quedan más elementos.



* *Función principal :* llamamos a la función cod\_primero con el primer elemento de L, de modo que Lfront sea la primera sublista de elementos repetidos. Recursivamente vamos llamando a cod\_all con Lrem, es decir, con lo que ha “sobrado” de la función para que realice lo mismo con el resto de elementos.



Hemos probado los siguientes ejemplos:

En este observamos que la lista devuelta es correcta y completa.

Este ejemplo nos muestra que al pasar el resultado nos devuelve True, es decir, corresponde a esa lista.

Este ejemplo lo prueba con solamente un elemento.

Finalmente, este ejemplo prueba que si pasamos una lista vacía, el resultado será una lista vacía pues no hay elementos repetidos

1. **RUN\_LENGHT(L, L1) :** este es el último apartado, en el que utilizamos las funciones anteriores para resolver el problema completo. Éste consiste en que, pasada una lista L de números repetidos como en los apartados anteriores, nos devuelve en L1, una lista de sublistas en las que cada una se compone del número de repeticiones del elemento, y el elemento que se repite.

* *Función first :* devuelve el primer elemento de una lista, la hemos implementado como función auxiliar para hacer más fácil la lectura del código.
* *Caso base :* las dos listas están vacías, es decir, hemos llegado al final del problema.



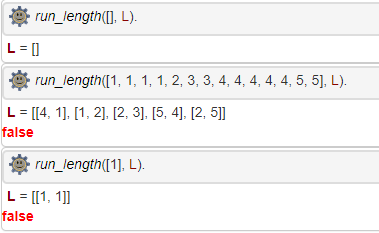
* *Función auxiliar :* hemos implementado una función auxiliar para la resolución del problema, ésta consiste en dividir la lista que devuelve cod\_all en la lista que necesitamos. Básicamente, por cada elemento de la lista que nos devuelve cod\_all, calcula su longitud y la une con el primer elemento de esa lista.



* *Función principal :* simplemente llama a cod\_all y comienza la recursión para dividir la lista resultada mediante la función auxiliar.



Hemos implementado los siguientes ejemplos:

Éste nos muestra que una lista vacía se concierte en otra vacía.

Este ejemplo prueba que funciona correctamente.

Este ejemplo es una prueba con un solo elemento.