VJ1203 – Programación I
Práctica 4: Funciones*
Curso 2014/15

CÓDIGO PARA PROBAR LOS PROGRAMAS QUE CONTIENEN FUNCIONES:

En el Aula Virtual encontraréis un fichero con código que os puede servir para probar el correcto funcionamiento de las funciones que escribáis. Básicamente es un módulo `modulo_test.py` con una sola función `test` que se puede utilizar en llamadas del tipo:

```
test(divisores(10) == [1,2,5,10])
```

La llamada anterior servirá para comprobar si nuestra función `divisores` se comporta como debería, devolviendo la lista `[1,2,5,10]` dado el número 10.

Tal y como se muestra en el fichero `divisores_test.py`, la forma de utilizar la función `test` en nuestros programas sería:

```
# -- Programa principal --
# Ejecutar el test sólo al ejecutar el fichero (y no al importarlo)
if __name__ == '__main__':
    # Código para ejecutar la función divisores con los datos
    # de prueba
    test(divisores(10) == [1,2,5,10])
```

Ejercicios con números

Ej. 1 — Números "narcisistas". Se denomina "narcisista" a todo número de n dígitos que es igual a la suma de sus dígitos elevados a la n-ésima potencia. Por ejemplo, el 153 es un número narcisista porque $153 = 1^3 + 5^3 + 3^3$.

Define una función `es_narcisista` que dado un número entero positivo devuelva un booleano que indique si el número es narcisista.

Ej. 2 — Calcular los divisores de un número. Define una función `divisores` que devuelva una lista con todos los divisores de un número entero positivo. Por ejemplo, dado el número 10 la función deberá devolver la lista `[1,2,5,10]`.

Ej. 3 — Números "amigos". Dos números son "amigos" si la suma de los divisores del primer número (excluido él) es igual al segundo número y viceversa, es decir, la

*La práctica 4 tiene una duración de dos sesiones.

suma de los divisores del segundo número (excluido él) es igual al primer número. Por ejemplo, 220 y 284 son amigos porque los divisores de 220 son 1, 2, 4, 5, 10, 11, 20, 22, 44, 55 y 110, que suman 284, y los divisores de 284 son 1, 2, 4, 71 y 142, que suman 220.

Tomando como punto de partida la función del **Ej. 2**, define una función `son_amigos` que dados dos números enteros positivos devuelva un booleano que indique si los números son amigos.

Ejercicios con cadenas

Ej. 4 — Calcular el contenido GC de una secuencia de ADN. Supondremos que se almacena en una cadena un fragmento de la secuencia de ADN del genoma de una persona. Dicha cadena contendrá únicamente los caracteres `''A''` (corresponde a adenina), `''C''` (citosina), `''G''` (guanina) y `''T''` (timina).

Define una función `contenido_GC`, que dada una secuencia de caracteres como la que se describe la recorra y devuelva el contenido GC de la misma, definido como el porcentaje de bases `''G''` y `''C''` que hay en la secuencia, redondeado a dos decimales. Por ejemplo, dadas las secuencias `''GCGC''` y `''CGCG''` la función deberá devolver el porcentaje 100,0. Y dada la secuencia `''GACGAC''` la función deberá devolver 66,67.

Ej. 5 — Calcular la expansión del triplete "CAG" de una secuencia de ADN. Define una función `expansion_triplete_CAG` que dada una cadena como la que se describe en el **Ej. 4** la recorra y devuelva el máximo número de repeticiones consecutivas del patrón `''CAG''`. Si el patrón no aparece en la cadena la función deberá devolver `None`. Por ejemplo, dada la secuencia `''GACGAC''` la función deberá devolver `None`, y dada la secuencia `''CAGCAG''` deberá devolver 2. Y con la secuencia `''CAGTACTACCAGCAGCAGCGCCAGCAGATCGA''` la función deberá devolver 3.
Nota: No se permite utilizar el operador de pertenencia.

Ej. 6 — Menú de operaciones sobre secuencias de ADN. Define una función `menu_ADN` que: (1) muestre por pantalla como opciones los cálculos sobre secuencias de ADN anteriores, (2) lea de teclado una opción, comprobando que es válida, y (3) devuelva el valor leído.

A continuación, escribe un programa que muestre el menú y pida al usuario una opción, y en función de la opción elegida lea de teclado los datos necesarios, llame a la función adecuada y muestre los resultados por pantalla.
Nota: Se deben importar las funciones de los ejercicios anteriores, no copiarlas.

Ejercicios con listas

Ej. 7 — Números "sociables". Los números "sociables" son una generalización del concepto de números amigos a una serie de tres o más números. Así, la suma de los divisores del primer número (excluido él) es igual al segundo, la suma de los divisores del segundo (excluido él) es igual al tercero, y así sucesivamente. Además, la suma de los divisores del último es igual al primer número de la lista.

Tomando como punto de partida la función del **Ej. 3**, define una función

`son_sociables` que dada una lista de números enteros positivos devuelva un booleano que indique si son sociables.

Ej. 8 — Construir una lista de dígitos a partir de un entero. Resuelve el problema del **Ej. 5 de la práctica 3** definiendo y utilizando una función `crear_lista_dígitos`, que dado un valor entero positivo construya la lista de dígitos correspondiente, que deberá devolver como resultado.

Después, escribe un programa que lea números de teclado y utilizando la función anterior muestre por pantalla la lista que lo representa, hasta que el usuario teclee un número negativo.

Ej. 9 — Sumar los enteros de dos listas de dígitos. Modifica el programa del **Ej. 7 de la práctica 3** para resolver el mismo problema definiendo y utilizando una función `sumar_listas_dígitos`, que dadas dos listas de dígitos las procese para obtener la lista con los dígitos del resultado de la suma, que deberá devolver como resultado. Si las listas no tienen el mismo número de dígitos la función deberá devolver `None`.

Ejercicios con matrices

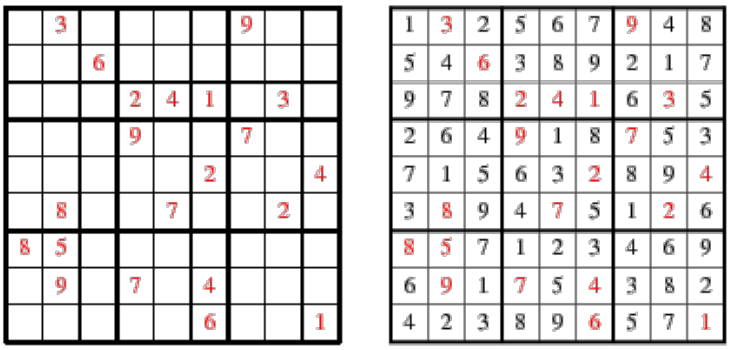
Ej. 10 — Leer de teclado/mostrar por pantalla una matriz de enteros. Define una función `leer_matriz_enteros` que sin necesidad de ningún parámetro construya una matriz de enteros positivos, leyendo los valores necesarios, es decir, el número de filas y de columnas, y cada uno de los valores de la matriz (para m filas y n columnas, $m \times n$ valores). La función deberá devolver la matriz creada como resultado. Define también una función `mostrar_matriz_enteros`, que dada una matriz de enteros positivos la muestre por pantalla en disposición bidimensional.

Después, escribe un programa que utilizando las funciones anteriores lea una matriz de teclado y a continuación la muestre por pantalla, repitiendo estos pasos mientras que el usuario indique que quiere continuar.

Ej. 11 — Determinar si una matriz es un cuadrado latino. Resolver el problema del **Ej. 5 de la práctica 3** definiendo y utilizando una función `es_cuadrado_latino`, que dada una matriz de enteros positivos devuelva un booleano que indique si la matriz es un cuadrado latino.

Ej. 12 — Determinar si una matriz es un "sudoku" (OPCIONAL). Un *"sudoku"* en su modalidad clásica es un caso especial de cuadrado latino. Se trata de un cuadrado latino porque es una matriz de dimensión 9×9 que solo contiene valores entre 1 y 9 y que cumple que en ninguna fila/columna se repite un valor. Además, las 9 regiones de dimensión 3×3 de las que se compone un sudoku contienen los valores del 1 al 9, sin repetir. Por ejemplo, la matriz de la siguiente figura es un sudoku:

(continua en la página siguiente)



Utilizando la función del **Ej. 11**, define una función `es_sudoku` que dada una matriz de números enteros positivos devuelva un booleano que indique si es un sudoku.
Nota: La función deberá llamar obligatoriamente a `es_cuadrado_latino`.