

NOMBRE Y APELLIDOS:

Aspectos generales de la prueba:

- Crear una carpeta en el escritorio denominada *examen_final*. Guarda todos los ejercicios en esa carpeta, uno archivo por cada ejercicio: *ejercicio1.py*, *ejercicio2.py*, *ejercicio3.py* y *ejercicio4.py*
- Al finalizar la prueba, subir los archivos por separado al directorio de moodle, situado al final de la página y denominado SUBIR EXAMEN FINAL 30/1/2017.

1. [50 %] Las piscinas de Morancas utilizan la tabla de tarifas siguiente:

EDAD	ENTRADA		BONO	
	— 10	11+	— 10	11 +
vecino	0	1'5	0	15
hijo del pueblo	0	2	0	26
forastero	0	2'5	0	47

Escribe un programa para la taquilla:

```
$ python taquilla.py
```

```
-----
----- PISCINAS MUNICIPALES -----
-----
```

```
Edad: 6
Tarifa gratuita
¿Entrada(E) o bono(B)? B
Nombre: Felipe Mediavilla
Generar el archivo (para impresión) bono.txt:
*****
PISCINAS MUNICIPALES
*****
BONO                               2017
Felipe Mediavilla

Hasta 6 años           Tarifa gratuita
*****
```

```
$ python taquilla.py
```

```
-----
----- PISCINAS MUNICIPALES -----
-----
```

```
Edad: 57
¿Entrada(E) o bono(B)? E
¿Vecino(V), h. pueblo(H) o forastero(F)? H
Son 2, a pagar en efectivo.
Importe recibido: 10
8 de vuelta
Generar el archivo (para impresión) entrada.txt:
*****
PISCINAS MUNICIPALES
*****
ENTRADA                               2017

Adulto                               Son 2 euros
*****
```

2. [20 %] El *doble factorial* $n!!$ de un número n se define como el producto de todos los números enteros positivos, no mayores que n y con la misma paridad de este número. Por ejemplo:

$$7!! = 1 \times 3 \times 5 \times 7, \quad 6!! = 2 \times 4 \times 6.$$

Escribe un programa (función) iterativo y otro recursivo para calcular el *doble factorial* de un entero positivo $n > 0$.

EJERCICIO CALIFICACIÓN RECUPERABLE

3. [30 %] Escribe dos programas en PYTHON que realicen respectivamente las siguientes dos tareas:

- Solicitar al usuario dos números n y m tales que $m > 5$, para calcular

$$\prod_{i=5}^m (n+i).$$

- Calcular el producto tensorial de dos vectores (=listas de longitud n) $\vec{u} = [a_1, a_2, \dots, a_n]$ y $\vec{v} = [b_1, b_2, \dots, b_n]$ solicitados al usuario.

Se define el producto tensorial de los vectores $\vec{u} \otimes \vec{v}$ como la lista, de longitud n , cuyos elementos son también listas de longitud n , determinadas como se describe a continuación:

$$\vec{u} \otimes \vec{v} = [[a_1 b_1, a_1 b_2, \dots, a_1 b_n], [a_2 b_1, a_2 b_2, \dots, a_2 b_n], \dots, [a_n b_1, a_n b_2, \dots, a_n b_n]]$$

Por ejemplo:

$$[2, 3] \otimes [1, 5] = [[2, 10], [3, 15]] \quad [1, 5, 0] \otimes [2, 7, 1] = [[2, 7, 1], [10, 35, 5], [0, 0, 0]]$$

EJERCICIO PARA SUBIR NOTA

4. Una versión del denominado **Problema de la Mochila** (The Knapsack Problem) consiste, dada una lista de números positivos $[a_1, a_2, \dots, a_n]$ tales que $a_1 < a_2 < \dots < a_n$ y un número s , decidir si existe un subconjunto de esa lista tales que la suma de sus elementos sea s .

Por ejemplo, si la lista es $[2, 7, 15, 34, 67]$ y s es 17, entonces un subconjunto es $\{2, 15\}$ puesto que $17 = 2 + 15$ y le representaremos por $[1, 0, 1, 0, 0]$, si $s = 43$ entonces un subconjunto es $[1, 1, 0, 1, 0]$ puesto que $43 = 2 + 7 + 34$, pero si s es 31 no existe ningún subconjunto de la lista cuya suma sea 31, y lo representaremos por -1 .

Este célebre problema modela llenar una mochila, incapaz de soportar más de un peso s determinado, con todo o parte de un conjunto de objetos, cada uno con un peso. Y es uno de los 21 problemas NP-completos (ver los apuntes de *algorítmica* de clase), establecidos por el informático Richard Karp en 1972.

Se pide programar en PYTHON una función `p_mochila` que tenga como entrada una lista de números creciente y un número s y devuelva un subconjunto de la lista cuya suma de elementos sea s , o devuelva -1 si no existe tal subconjunto.

```
>>>p_mochila([2,7,15,34,67],17)
[1,0,1,0,0]
>>>p_mochila([2,7,15,34,67],43)
[1,1,0,1,0]
>>>p_mochila([2,7,15,34,67],31)
-1
```
