

Este boletín de ejercicios está orientado a practicar con

- programación modular
- gestión de excepciones
- expresiones regulares
- ejercicios de examen

1. Queremos escribir un programa que nos permita esconder un número pin de cuatro cifras por un método muy sencillo. Imagínate que tu pin es el 6240. La salida por consola sería así:

```
XXXXX0XXXX
X0XXXXXXXXX
XXX0XXXXXXX
XXXXXXXXXX0
```

Escribe una función que **reciba como argumento un pin numérico de 4 cifras** (por ejemplo 6420) y **devuelva una estructura con cuatro elementos** cada uno de los cuales es una línea. Luego, muestra en consola el resultado tal y como se ve aquí arriba. La función debe de ser exactamente como se describe: el pin se recibe como un argumento numérico. La función devuelve una estructura (no la muestra) y hay que mostrarla como aquí arriba desde fuera de la función.

NOTAS: Fíjate que el 0 se corresponde con la última posición y no con la primera

Piensa que los primeros dígitos del pin pueden ser 0 (ejemplo: 125, 40 o 7) y que como el argumento de la función que se pide es un entero no puedes escribir 0040, pero tu función debe de funcionar también correctamente en estos casos

Se valorará si haces el desarrollo de forma modular usando una función que reciba un entero y devuelva la línea correspondiente. Eso permitiría en un futuro modificar el método de cifrado fácilmente

2. Queremos ahora cambiar el sistema de codificación. Con el mismo pin (6240) la salida ahora sería esta:

```
XXXX000000
XXXXXXXXX00
XXXXXX0000
XXXXXXXXXXX
```

Escribe la función que lo realice. Ten en cuenta las mismas consideraciones que en el ejercicio anterior

3. Desarrolla una función que reciba como argumento un número en binario (como una cadena de texto o String) y devuelva su valor en decimal. A continuación tienes unos ejemplos de ejecución y la salida que producen en consola

```
print(toDecimal("10110"))  
print(toDecimal("345"))  
print(toDecimal("hola"))
```

Salida en consola:

```
22  
-1  
-1
```

La función debe de ser exactamente como se describe: el número en binario se recibe como argumento y como string. El valor en decimal se devuelve (y no se imprime desde dentro de la función). Si lo que se recibe no es un número en binario, la función debe de devolver el valor -1

4. Desarrolla una función que reciba como argumento un número en decimal positivo y menor o igual a 255 y devuelva su valor en binario como un octeto completo. A continuación tienes unos ejemplos de ejecución y la salida que producen en consola

```
print(toBinario(22))  
print(toBinario(129))  
print(toBinario(345))  
print(toBinario(22.5))  
print(toBinario("hola"))  
print(toBinario(-2))
```

Salida en consola:

```
00010110  
10000001  
-1  
-1  
-1  
-1
```

La función debe de ser exactamente como se describe: el número en decimal se recibe como argumento. El valor en binario se devuelve (y no se imprime desde dentro de la función) como String (si no los ceros a la izquierda desaparecerían) y como un octeto completo. Si lo que se recibe no es válido (un string, un número decimal, negativo o superior a 255) tu función devolverá un -1

5. Escribe una función que reciba **dos números enteros, positivos y mayores a cero** y te calcule los divisores comunes que tienen ambos. A continuación tienes unos ejemplos de ejecución y la salida que producen en consola

```
divisoresComunes(22, 16)
divisoresComunes(33, 17)
divisoresComunes(1725, 2500)
```

Salida en consola:

```
Los divisores comunes de 22 y 16 son: 1, 2
El único divisor común de 33 y 17 es: 1
Los divisores comunes de 1725 y 2500 son: 1, 5, 25
```

Si alguno de los argumentos no cumple con lo dicho aquí arriba, tu función debería de mostrar un mensaje de error:

```
divisoresComunes(22.5, 0)
```

```
No puedo calcular los divisores comunes de esos números
```

NOTAS: Se valorará si haces el desarrollo de forma modular usando una función que reciba un número y devuelva todos sus divisores en el formato que creas oportuno.

Fíjate que tu función debería de distinguir en la salida el caso en que el divisor común es únicamente el 1

Los divisores comunes deberían de mostrarse ordenados de forma ascendente

6. Sabrás, por las clases de Sistemas de la Información, que, una IP de la versión 4 está formada por cuatro bytes y que en relación a su primer byte podemos saber de que clase es y su máscara (si estamos usando direccionamiento sin clases):

- En la clase A el primer byte tiene que estar entre el 0 y el 127, ambos inclusive y la máscara es /8
- En la clase B el primer byte tiene que estar entre el 128 y el 191, ambos inclusive y la máscara es /16
- En la clase C el primer byte tiene que estar entre el 192 y el 223, ambos inclusive y su máscara es /24
- Entre el 224 y el 255, ambos inclusive, las direcciones se consideran reservadas (clases D y E)
- Por encima del 255 no son válidas

Escribe un programa que te pida una dirección IP sin máscara por el teclado y te la escriba en consola poniéndole la máscara adecuada. Un ejemplo de ejecución podría ser así:

**Introduce una dirección IP: 8.8.8.8
8.8.8.8/8**

Si la dirección IP introducida es reservada o no válida tu programa debería de darte un mensaje por pantalla :

**Introduce una dirección IP: 230.0.0.0
Dirección reservada**

**Introduce una dirección IP: 300.1.1.1
Dirección no válida**

**Introduce una dirección IP: OLAKASE
Dirección no válida**

NOTA: Tu programa debería de contemplar también, a la hora de decir que una IP no es válida, que no tenga el formato adecuado (cuatro números enteros positivos separados por puntos y nunca superiores al 255)

7. Escribe **una función que reciba un como argumento un número variable de strings** y nos diga, por cada uno de ellos si se trata de una dirección MAC válida o no. Las direcciones MAC, por si no lo recuerdas de las clases de sistemas de información, están formadas por seis bytes expresados en hexadecimal. Por ejemplo así:

F4:8E:38:AF:F4:1C

Cada cifra en hexadecimal no puede ser mayor que 255, es decir, de FF. Tú función debe de contemplar que las letras correspondientes a las cifras A-F del hexadecimal pueden venir en mayúsculas o en minúsculas y debería de darlas como válidas en ambos casos. También que a veces se escribe separando cada byte de forma independiente con dos puntos (como en el ejemplo anterior) o en el formato que usa CISCO, separando cada dos bytes con un punto. Así:

F48E.38AF.F41C

A continuación dos ejemplos de llamada a la función con sus respectivas salidas en consola:

macsValidas("F4:8E:38:AF:F4:1C", "7521-MXP")

**F4:8E:38:AF:F4:1C es válida
7521-MXP no es válida**

**MACs válidas: 1
MACs no válidas: 1**

macsValidas("F48E38AFF41C", "f48e.38af.f41c", "F:3:AF:4:1:11")

F48E38AFF41C no es válida
f48e.38af.f41c es válida
F:3:AF:4:1:11 no es válida

MACs válidas: 1
MACs no válidas: 2

NOTA: Se valorará si haces el desarrollo de forma modular usando una función que reciba una MAC y devuelva un valor booleano diciendo si es válida o no.

8. Escribe una función que reciba un como argumento un número variable de strings y nos diga, por cada uno de ellos, si se trata de una matrícula de coche válida.

- La matrícula debe de estar formada por cuatro dígitos y tres letras
- Las letras Ñ, Q o cualquiera de las vocales no son válidas

Serían, por ejemplo, matrículas inválidas: "22CDR", "3456BAC" o "1224MN" y serían matrículas válidas "2222CDR", "3456BBC" o "1224MNP"

Tú función debe de contemplar que la matrícula puede venir con las letras en mayúsculas o en minúsculas y debería de darla como válida. También que a veces se escribe separando los números de las letras con un espacio (pero solo uno) o un guión y también debería de considerarse válido.

Serían válidas, por ejemplo, "7521-MXP", "6555KFG" o "5432 BCF" pero no lo serían "7521 MXP" (con cinco espacios entre números y letras) o "5432 – BCF" (con un espacio, un guión y un espacio entre medios). Es decir: si hay algo entre los números y las letras solo se admite que sea un único espacio o un único guión.

A continuación dos ejemplos de llamada a la función con sus respectivas salidas en consola:

matriculasValidas("22CDR", "7521-MXP" , "1224MN")

22CDR no es válida
7521-MXP es válida
1224MN no es válida

Matrículas válidas: 1
Matrículas no válidas: 2

matriculasValidas("5432 - BCF", "3456BAC")

5432 - BCF no es válida

3456BAC no es válida

Matrículas válidas: 0

Matrículas no válidas: 2

NOTA: Puedes hacer el ejercicio usando expresiones regulares o no, pero la nota máxima del mismo solo se obtiene si las usas.

Se valorará también si haces el desarrollo de forma modular usando una función que reciba una matrícula y devuelva un valor booleano diciendo si es válida o no.