

- **65** Diseña un programa que, dado un número entero, determine si éste es el doble de un número impar. (Ejemplo: 14 es el doble de 7, que es impar.)
- **66** Diseña un programa que, dados dos números enteros, muestre por pantalla uno de estos mensajes: «El segundo es el cuadrado exacto del primero.», «El segundo es menor que el cuadrado del primero.» o «El segundo es mayor que el cuadrado del primero.», dependiendo de la verificación de la condición correspondiente al significado de cada mensaje.
- **67** Un capital de C euros a un interés del x por cien anual durante n años se convierte en $C \cdot (1 + x/100)^n$ euros. Diseña un programa Python que solicite la cantidad C y el interés x y calcule el capital final *sólo si x es una cantidad positiva*.
- **68** Realiza un programa que calcule el desglose en billetes y monedas de una cantidad exacta de euros. Hay billetes de 500, 200, 100, 50, 20, 10 y 5 € y monedas de 2 y 1 €.

Por ejemplo, si deseamos conocer el desglose de 434 €, el programa mostrará por pantalla el siguiente resultado:

```
2 billetes de 200 euros.
1 billete de 20 euros.
1 billete de 10 euros.
2 monedas de 2 euros.
```

(¿Que cómo se efectúa el desglose? Muy fácil. Empieza por calcular la división entera entre la cantidad y 500 (el valor de la mayor moneda): 434 entre 500 da 0, así que no hay billetes de 500 € en el desglose; divide a continuación la cantidad 434 entre 200, cabe a 2 y sobran 34, así que en el desglose hay 2 billetes de 200 €; dividimos a continuación 34 entre 100 y vemos que no hay ningún billete de 100 € en el desglose (cabe a 0); como el resto de la última división es 34, pasamos a dividir 34 entre 20 y vemos que el desglose incluye un billete de 20 € y aún nos faltan 14 € por desglosar...)


- **69** ¿Hay alguna diferencia entre el programa anterior y este otro cuando los ejecutamos?

```
segundo_grado.3.py  segundo_grado.py
1 from math import sqrt
2
3 a = float(raw_input('Valor de a: '))
4 b = float(raw_input('Valor de b: '))
5 c = float(raw_input('Valor de c: '))
6
7 if a == 0:
8     if b == 0:
9         if c == 0:
10             print 'La ecuación tiene infinitas soluciones.'
11         else:
12             print 'La ecuación no tiene solución.'
13     else:
14         x = -c / b
15         print 'Solución de la ecuación: x=%4.3f' % x
16 else:
17     x1 = (-b + sqrt(b**2 - 4*a*c)) / (2 * a)
18     x2 = (-b - sqrt(b**2 - 4*a*c)) / (2 * a)
19     print 'Soluciones de la ecuación: x1=%4.3f y x2=%4.3f' % (x1, x2)
```


- **70** ¿Hay alguna diferencia entre el programa anterior y este otro cuando los ejecutamos?

```
segundo_grado.4.py  segundo_grado.py
1 from math import sqrt
2
3 a = float(raw_input('Valor de a: '))
4 b = float(raw_input('Valor de b: '))
5 c = float(raw_input('Valor de c: '))
6
7 if a == 0 and b == 0 and c == 0:
8     print 'La ecuación tiene infinitas soluciones.'
9 else:
10     if a == 0 and b == 0:
11         print 'La ecuación no tiene solución.'
12     else:
13         if a == 0:
14             x = -c / b
15             print 'Solución de la ecuación: x=%4.3f' % x
```


- **81** Diseña un programa que calcule la menor de cinco palabras dadas; es decir, la primera palabra de las cinco en orden alfabético. Aceptaremos que las mayúsculas son «alfabéticamente» menores que las minúsculas, de acuerdo con la tabla ASCII.
- **82** Diseña un programa que calcule la menor de cinco palabras dadas; es decir, la primera palabra de las cinco en orden alfabético. *No* aceptaremos que las mayúsculas sean «alfabéticamente» menores que las minúsculas. O sea, 'pepita' es menor que 'Pepito'.
- **83** Diseña un programa que, dados cinco números enteros, determine cuál de los cuatro últimos números es más cercano al primero. (Por ejemplo, si el usuario introduce los números 2, 6, 4, 1 y 10, el programa responderá que el número más cercano al 2 es el 1.)
- **84** Diseña un programa que, dados cinco puntos en el plano, determine cuál de los cuatro últimos puntos es más cercano al primero. Un punto se representará con dos variables: una para la abscisa y otra para la ordenada. La distancia entre dos puntos (x_1, y_1) y (x_2, y_2) es $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.
- **85** Indica en cada uno de los siguientes programas qué valores o rangos de valores provocan la aparición de los distintos mensajes:

- a)  `aparcar.py` `aparcar.py`
- ```


1 dia = int(raw_input('Dime qué día es hoy: '))
2
3 if 0 < dia <= 15:
4 print 'Puedes aparcar en el lado izquierdo de la calle'
5 else:
6 if 15 < dia < 32:
7 print 'Puedes aparcar en el lado derecho de la calle'
8 else:
9 print 'Ningún mes tiene %d días.' % dia

```
- b)  `estaciones.py` `estaciones.py`
- ```

1 mes = int(raw_input('Dame un mes: '))
2
3 if 1 <= mes <= 3:
4     print 'Invierno.'
5 else:
6     if mes == 4 or mes == 5 or mes == 6:
7         print 'Primavera.'
8     else:
9         if not (mes < 7 or 9 < mes):
10            print 'Verano.'
11        else:
12            if not (mes != 10 and mes != 11 and mes != 12):
13                print 'Otoño.'
14            else:
15                print 'Ningún año tiene %d meses.' % mes

```
- c)  `identificador.py` `identificador.py`
- ```

1 car = raw_input('Dame un carácter: ')
2
3 if 'a' <= car.lower() <= 'z' or car == '_':
4 print 'Este carácter es válido en un identificador en Python.'
5 else:
6 if not (car < '0' or '9' < car):
7 print 'Un dígito es válido en un identificador en Python,',
8 print 'siempre que no sea el primer carácter.'
9 else:
10 print 'Carácter no válido para formar un identificador en Python.'

```
- d)  `bisiesto.py` `bisiesto.py`
- ```

1 anyo = int(raw_input('Dame un año: '))
2
3 if anyo % 4 == 0 and (anyo % 100 != 0 or anyo % 400 == 0):
4     print 'El año %d es bisiesto.' % anyo
5 else:
6     print 'El año %d no es bisiesto.' % anyo

```

ejercicio_bucle.16.py

ejercicio_bucle.py

```

1 i = int(raw_input('Valor_inicial:_'))
2 limite = int(raw_input('Límite:_'))
3 incremento = int(raw_input('Incremento:_'))
4 while i < limite:
5     print i
6     i += incremento

```

- **103** Implementa un programa que muestre todos los múltiplos de 6 entre 6 y 150, ambos inclusive.
- **104** Implementa un programa que muestre todos los múltiplos de n entre n y $m \cdot n$, ambos inclusive, donde n y m son números introducidos por el usuario.
- **105** Implementa un programa que muestre todos los números potencia de 2 entre 2^0 y 2^{30} , ambos inclusive.
- **106** Estudia las diferencias entre el siguiente programa y el último que hemos estudiado. ¿Producen ambos el mismo resultado?

sumatorio.2.py

sumatorio.py

```

1 sumatorio = 0
2 i = 0
3 while i < 1000:
4     i += 1
5     sumatorio += i
6 print sumatorio

```

- **107** Diseña un programa que calcule

$$\sum_{i=n}^m i,$$

donde n y m son números enteros que deberá introducir el usuario por teclado.

- **108** Modifica el programa anterior para que si $n > m$, el programa no efectúe ningún cálculo y muestre por pantalla un mensaje que diga que n debe ser menor o igual que m .
- **109** Queremos hacer un programa que calcule el factorial de un número entero positivo. El factorial de n se denota con $n!$, pero no existe ningún operador Python que permita efectuar este cálculo directamente. Sabiendo que

$$n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot (n-1) \cdot n$$

y que $0! = 1$, haz un programa que pida el valor de n y muestre por pantalla el resultado de calcular $n!$.

- **110** El número de combinaciones que podemos formar tomando m elementos de un conjunto con n elementos es:

$$C_n^m = \binom{n}{m} = \frac{n!}{(n-m)!m!}.$$

Diseña un programa que pida el valor de n y m y calcule C_n^m . (Ten en cuenta que n ha de ser mayor o igual que m .)
(Puedes comprobar la validez de tu programa introduciendo los valores $n = 15$ y $m = 10$: el resultado es 3003.)

- **111** ¿Qué te parece esta otra versión del mismo programa?

raiz.2.py

raiz.py

```

1 from math import sqrt
2
3 x = float(raw_input('Introduce_un_número_positivo:_'))
4 while x < 0:
5     x = float(raw_input('Introduce_un_número_positivo:_'))
6
7 print 'La_raíz_cuadrada_de_%f_es_%f' % (x, sqrt(x))

```

- **112** Diseña un programa que solicite la lectura de un número entre 0 y 10 (ambos inclusive). Si el usuario teclea un número fuera del rango válido, el programa solicitará nuevamente la introducción del valor cuantas veces sea menester.
- **113** Diseña un programa que solicite la lectura de un texto que no contenga letras mayúsculas. Si el usuario teclea una letra mayúscula, el programa solicitará nuevamente la introducción del texto cuantas veces sea preciso.

- **184** ¿Qué pasa si introducimos una cadena con caracteres que no pertenecen al conjunto de dígitos binarios como, por ejemplo, '101a2'? Modifica el programa para que, en tal caso, muestre en pantalla el mensaje «Número binario mal formado» y solicite nuevamente la introducción de la cadena.
- **185** Diseña un programa que convierta una cadena de dígitos entre el «0» y el «7» al valor correspondiente a una interpretación de dicha cadena como número en base octal.
- **186** Diseña un programa que convierta una cadena de dígitos o letras entre la «a» y la «f» al valor correspondiente a una interpretación de dicha cadena como número en base hexadecimal.
- **187** Diseña un programa que reciba una cadena que codifica un número en octal, decimal o hexadecimal y muestre el valor de dicho número. Si la cadena empieza por «0x» o «0X» se interpretará como un número hexadecimal (ejemplo: '0xff' es 255); si no, si el primer carácter es «0», la cadena se interpretará como un número octal (ejemplo: '017' es 15); y si no, se interpretará como un número decimal (ejemplo: '99' es 99).
- **188** Diseña un programa que lea un número entero y muestre una cadena con su representación octal.
- **189** Diseña un programa que lea una cadena que representa un número codificado en base 8 y muestre por pantalla su representación en base 2.
- **190** Una palabra es «alfabética» si todas sus letras están ordenadas alfabéticamente. Por ejemplo, «amor», «chino» e «himno» son palabras «alfabéticas». Diseña un programa que lea una palabra y nos diga si es alfabética o no.
- **191** Diseña un programa que nos diga si una cadena es palíndromo o no. Una cadena es palíndromo si se lee igual de izquierda a derecha que de derecha a izquierda. Por ejemplo, 'ana' es un palíndromo.
- **192** Una frase es palíndromo si se lee igual de derecha a izquierda que de izquierda a derecha, pero obviando los espacios en blanco y los signos de puntuación. Por ejemplo, las cadenas 'sé₁verla₂al₃revés', 'anita₁lava₂la₃ Tina', 'luz₁azul' y 'la₁ruta₂natural' contienen frases palíndromas. Diseña un programa que diga si una frase es o no es palíndroma.
- **193** Probablemente el programa que has diseñado para el ejercicio anterior falle ante frases palíndromas como éstas: «Dábale arroz a la zorra el abad», «Salta Lenín el atlas», «Amigo, no gima», «Átale, demoníaco Caín, o me delata», «Anás usó tu auto, Susana», «A Mercedes, ése de crema», «A mamá Roma le aviva el amor a papá, y a papá Roma le aviva el amor a mamá» y «¡arriba la birra!», pues hemos de comparar ciertas letras con sus versiones acentuadas, o mayúsculas o la apertura de exclamación con su cierre. Modifica tu programa para que identifique correctamente frases palíndromas en las que pueden aparecer letras mayúsculas, vocales acentuadas y la vocal «u» con diéresis.
- **194** Hay un tipo de pasatiempos que propone descifrar un texto del que se han suprimido las vocales. Por ejemplo, el texto «.n .j.mpl. d. p.s.t..mp.s», se descifra sustituyendo cada punto con una vocal del texto. La solución es «un ejemplo de pasatiempos». Diseña un programa que ayude al creador de pasatiempos. El programa recibirá una cadena y mostrará otra en la que cada vocal ha sido reemplazada por un punto.
- **195** El nombre de un fichero es una cadena que puede tener lo que denominamos una extensión. La extensión de un nombre de fichero es la serie de caracteres que suceden al último punto presente en la cadena. Si el nombre no tiene ningún punto, asumiremos que su extensión es la cadena vacía. Haz un programa que solicite el nombre de un fichero y muestre por pantalla los caracteres que forman su extensión. Prueba la validez de tu programa pidiendo que muestre la extensión de los nombres de fichero documento.doc y tema.1.tex, que son doc y tex, respectivamente.
- **196** Haz un programa que lea dos cadenas que representen sendos números binarios. A continuación, el programa mostrará el número binario que resulta de sumar ambos (y que será otra cadena). Si, por ejemplo, el usuario introduce las cadenas '100' y '111', el programa mostrará como resultado la cadena '1011'.
- (Nota: El procedimiento de suma con acarreo que implementes deberá trabajar directamente con la representación binaria leída.)
- **197** Una de las técnicas de criptografía más rudimentarias consiste en sustituir cada uno de los caracteres por otro situado n posiciones más a la derecha. Si $n = 2$, por ejemplo, sustituiremos la «a» por la «c», la «b» por la «e», y así sucesivamente. El problema que aparece en las últimas n letras del alfabeto tiene fácil solución: en el ejemplo, la letra «y» se sustituirá por la «a» y la letra «z» por la «b». La sustitución debe aplicarse a las letras minúsculas y mayúsculas y a los dígitos (el «0» se sustituye por el «2», el «1» por el «3» y así hasta llegar al «9», que se sustituye por el «1»).
- Diseña un programa que lea un texto y el valor de n y muestre su versión criptografiada.
- **198** Diseña un programa que lea un texto criptografiado siguiendo la técnica descrita en el apartado anterior y el valor de n utilizado al encriptar para mostrar ahora el texto decodificado.
- **199** ¿Y si se introduce un valor de i negativo? Corrige el programa para que detecte esa posibilidad e interprete un índice inicial negativo como el índice 0.
- **200** ¿No será también problemático que introduzcamos un valor del índice i mayor o igual que el de j ? ¿Se producirá entonces un error de ejecución? ¿Por qué?

- **263** Define una función llamada *area_circulo* que, a partir del radio de un círculo, devuelva el valor de su área. Utiliza el valor 3.1416 como aproximación de π o importa el valor de π que encontrarás en el módulo *math*.
(Recuerda que el área de un círculo es πr^2 .)
- **264** Define una función que convierta grados Fahrenheit en grados centígrados.
(Para calcular los grados centígrados has de restar 32 a los grados Fahrenheit y multiplicar el resultado por cinco novenos.)
- **265** Define una función que convierta grados centígrados en grados Fahrenheit.
- **266** Define una función que convierta radianes en grados.
(Recuerda que 360 grados son 2π radianes.)
- **267** Define una función que convierta grados en radianes.
- **268** ¿Es este programa equivalente al que acabamos de ver?

```

mayoria.edad.3.py
1 def mayoria_de_edad(edad):
2     if edad < 18:
3         return False
4     return True

```

- **269** ¿Es este programa equivalente al que acabamos de ver?

```

mayoria.edad.4.py
1 def mayoria_de_edad(edad):
2     return edad >= 18

```

- **270** La última letra del DNI puede calcularse a partir del número. Para ello sólo tienes que dividir el número por 23 y quedarte con el resto, que es un número entre 0 y 22. La letra que corresponde a cada número la tienes en esta tabla:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
T	R	W	A	G	M	Y	F	P	D	X	B	N	J	Z	S	Q	V	H	L	C	K	E

Define una función que, dado un número de DNI, devuelva la letra que le corresponde.

- **271** Diseña una función que reciba una cadena y devuelva cierto si empieza por minúscula y falso en caso contrario.
- **272** Diseña una función llamada *es_repeticion* que reciba una cadena y nos diga si la cadena está formada mediante la concatenación de una cadena consigo misma. Por ejemplo, *es_repeticion('abab')* devolverá *True*, pues la cadena 'abab' está formada con la cadena 'ab' repetida; por contra *es_repeticion('ababab')* devolverá *False*.
- **273** ¿En qué se ha equivocado nuestro aprendiz de programador al escribir esta función?

```

perfecto.2.py
1 def es_perfecto(n):
2     for i in range(1, n):
3         sumatorio = 0
4         if n % i == 0:
5             sumatorio += i
6     return sumatorio == n

```

- **274** Mejora la función *es_perfecto* haciéndola más rápida. ¿Es realmente necesario considerar todos los números entre 1 y $n-1$?
- **275** Diseña una función que devuelva una lista con los números perfectos comprendidos entre 1 y n , siendo n un entero que nos proporciona el usuario.
- **276** Define una función que devuelva el número de días que tiene un año determinado. Ten en cuenta que un año es bisiesto si es divisible por 4 y no divisible por 100, excepto si es también divisible por 400, en cuyo caso es bisiesto.
(Ejemplos: El número de días de 2002 es 365: el número 2002 no es divisible por 4, así que no es bisiesto. El año 2004 es bisiesto y tiene 366 días: el número 2004 es divisible por 4, pero no por 100, así que es bisiesto. El año 1900 es divisible por 4, pero no es bisiesto porque es divisible por 100 y no por 400. El año 2000 sí es bisiesto: el número 2000 es divisible por 4 y, aunque es divisible por 100, también lo es por 400.)
- **277** Diseña una función que calcule el sumatorio de la diferencia entre números contiguos en una lista. Por ejemplo, para la lista [1, 3, 6, 10] devolverá 9, que es $2 + 3 + 4$ (el 2 resulta de calcular $3 - 1$, el 3 de calcular $6 - 3$ y el 4 de calcular $10 - 6$).
¿Sabes efectuar el cálculo de ese sumatorio sin utilizar bucles (ni la función *sum*)?
- **278** Haz una traza de la llamada *maximo*([6, 2, 7, 1, 10, 1, 0]).

► **279** Diseña una función que, dada una lista de números enteros, devuelva el número de «series» que hay en ella. Llamamos «serie» a todo tramo de la lista con valores idénticos.

Por ejemplo, la lista [1, 1, 8, 8, 8, 8, 0, 0, 0, 2, 10, 10] tiene 5 «series» (ten en cuenta que el 2 forma parte de una «serie» de un solo elemento).

► **280** Diseña una función que diga en qué posición empieza la «serie» más larga de una lista. En el ejemplo del ejercicio anterior, la «serie» más larga empieza en la posición 2 (que es el índice donde aparece el primer 8). (Nota: si hay dos «series» de igual longitud y ésta es la mayor, debes devolver la posición de la primera de las «series». Por ejemplo, para [8, 2, 2, 9, 9] deberás devolver la posición 1.)

► **281** Haz una función que reciba una lista de números y devuelva la media de dichos números. Ten cuidado con la lista vacía (su media es cero).

► **282** Diseña una función que calcule el productorio de todos los números que componen una lista.

► **283** Diseña una función que devuelva el valor absoluto de la máxima diferencia entre dos elementos consecutivos de una lista. Por ejemplo, el valor devuelto para la lista [1, 10, 2, 6, 2, 0] es 9, pues es la diferencia entre el valor 1 y el valor 10.

► **284** Diseña una función que devuelva el valor absoluto de la máxima diferencia entre cualquier par de elementos de una lista. Por ejemplo, el valor devuelto para la lista [1, 10, 2, 6, 8, 20] es 9, pues es la diferencia entre el valor 10 y el valor 0. (Pista: te puede convenir conocer el valor máximo y el valor mínimo de la lista.)

► **285** Modifica la función del ejercicio anterior para que devuelva el valor 0 tan pronto encuentre un 0 en la lista.

► **286** Define una función que, dada una cadena x , devuelva otra cuyo contenido sea el resultado de concatenar 6 veces x consigo misma.

► **287** Diseña una función que, dada una lista de cadenas, devuelva la cadena más larga. Si dos o más cadenas miden lo mismo y son las más largas, la función devolverá una cualquiera de ellas.

(Ejemplo: dada la lista ['Pepe', 'Juan', 'María', 'Ana'], la función devolverá la cadena 'María'.)

► **288** Diseña una función que, dada una lista de cadenas, devuelva una lista con todas las cadenas más largas, es decir, si dos o más cadenas miden lo mismo y son las más largas, la lista las contendrá a todas.

(Ejemplo: dada la lista ['Pepe', 'Ana', 'Juan', 'Paz'], la función devolverá la lista de dos elementos ['Pepe', 'Juan'].)

► **289** Diseña una función que reciba una lista de cadenas y devuelva el prefijo común más largo. Por ejemplo, la cadena 'pol' es el prefijo común más largo de esta lista:

['poliedro', 'policía', 'polífona', 'polinizar', 'polaridad', 'política']

► **290** Define una función que, dado el valor de los tres lados de un triángulo, devuelva la longitud de su perímetro.

► **291** Define una función que, dados dos parámetros b y x , devuelva el valor de $\log_b(x)$, es decir, el logaritmo en base b de x .

► **292** Diseña una función que devuelva la solución de la ecuación lineal $ax + b = 0$ dados a y b . Si la ecuación tiene infinitas soluciones o no tiene solución alguna, la función lo detectará y devolverá el valor *None*.

► **293** Diseña una función que calcule $\sum_{i=a}^b i$ dados a y b . Si a es mayor que b , la función devolverá el valor 0.

► **294** Diseña una función que calcule $\prod_{i=a}^b i$ dados a y b . Si a es mayor que b , la función devolverá el valor 0. Si 0 se encuentra entre a y b , la función devolverá también el valor cero, pero sin necesidad de iterar en un bucle.

► **295** Define una función llamada *raiz_n_esima* que devuelva el valor de $\sqrt[n]{x}$. (Nota: recuerda que $\sqrt[n]{x}$ es $x^{1/n}$).

► **296** Haz una función que reciba un número de DNI y una letra. La función devolverá *True* si la letra corresponde a ese número de DNI, y *False* en caso contrario. La función debe llamarse *comprueba_letra_dni*.

Si lo deseas, puedes llamar a la función *letra_dni*, desarrollada en el ejercicio 270, desde esta nueva función.

► **297** Diseña una función que diga (mediante la devolución de *True* o *False*) si dos números son *amigos*. Dos números son amigos si la suma de los divisores del primero (excluido él) es igual al segundo y viceversa.

► **298** ¿Funciona esta otra versión de *menu*?

function_menu.2.py

function_menu.py

```
1 def menu():
2     opcion = ''
3     while len(opcion) != 1 or opcion not in 'abc':
4         print 'Cajero_automático.'
5         print 'a)_Ingresar_dinero.'
6         print 'b)_Sacar_dinero.'
```



```

7     print 'c)_Consultar_saldo.'
8     opcion = raw_input('Escoja una opción:')
9     if len(opcion) != 1 or opcion not in 'abc':
10        print 'Sólo puede escoger las letras a, b o c. Inténtelo de nuevo.'
11    return opcion

```

► **299** Diseña una función llamada *menu_generico* que reciba una lista con opciones. Cada opción se asociará a un número entre 1 y la talla de la lista y la función mostrará por pantalla el menú con el número asociado a cada opción. El usuario deberá introducir por teclado una opción. Si la opción es válida, se devolverá su valor, y si no, se le advertirá del error y se solicitará nuevamente la introducción de un valor.

He aquí un ejemplo de llamada a la función:

```
menu_generico(['Saludar', 'Despedirse', 'Salir'])
```

Al ejecutarla, obtendremos en pantalla el siguiente texto:

```

1) Saludar
2) Despedirse
3) Salir
Escoja opción:

```

► **300** En un programa que estamos diseñando preguntamos al usuario numerosas cuestiones que requieren una respuesta afirmativa o negativa. Diseña una función llamada *si_o_no* que reciba una cadena (la pregunta). Dicha cadena se mostrará por pantalla y se solicitará al usuario que responda. Sólo aceptaremos como respuestas válidas 'si', 's', 'Si', 'SI', 'no', 'n', 'No', 'NO', las cuatro primeras para respuestas afirmativas y las cuatro últimas para respuestas negativas. Cada vez que el usuario se equivoque, en pantalla aparecerá un mensaje que le recuerde las respuestas aceptables. La función devolverá *True* si la respuesta es afirmativa, y *False* en caso contrario.

► **301** Diseña una función sin argumentos que devuelva un número aleatorio mayor o igual que 0.0 y menor que 10.0. Puedes llamar a la función *random* desde tu función.

► **302** Diseña una función sin argumentos que devuelva un número aleatorio mayor o igual que -10.0 y menor que 10.0.

► **303** Para diseñar un juego de tablero nos vendrá bien disponer de un «dado electrónico». Escribe una función Python sin argumentos llamada *dado* que devuelva un número *entero* aleatorio entre 1 y 6.

► **304** Diseña un programa que, dado un número *n*, muestre por pantalla todas las parejas de números amigos menores que *n*. La impresión de los resultados debe hacerse desde un procedimiento.

Dos números amigos sólo deberán aparecer una vez por pantalla. Por ejemplo, 220 y 284 son amigos: si aparece el mensaje «220 y 284 son amigos», no podrá aparecer el mensaje «284 y 220 son amigos», pues es redundante.

Debes diseñar una función que diga si dos números son amigos y un procedimiento que muestre la tabla.

► **305** Implementa un procedimiento Python tal que, dado un número entero, muestre por pantalla sus cifras en orden inverso. Por ejemplo, si el procedimiento recibe el número 324, mostrará por pantalla el 4, el 2 y el 3 (en líneas diferentes).

► **306** Diseña una función *es_primo* que determine si un número es primo (devolviendo *True*) o no (devolviendo *False*). Diseña a continuación un procedimiento *muestra_primos* que reciba un número y muestre por pantalla todos los números primos entre 1 y dicho número.

► **307** En el problema de los alumnos y las notas, se pide:

- Diseñar un *procedimiento* que reciba las dos listas y muestre por pantalla el nombre de todos los estudiantes que aprobaron el examen.
- Diseñar una *función* que reciba la lista de notas y devuelva el número de aprobados.
- Diseñar un *procedimiento* que reciba las dos listas y muestre por pantalla el nombre de todos los estudiantes que obtuvieron la máxima nota.
- Diseñar un *procedimiento* que reciba las dos listas y muestre por pantalla el nombre de todos los estudiantes cuya calificación es igual o superior a la calificación media.
- Diseñar una *función* que reciba las dos listas y un nombre (una cadena); si el nombre está en la lista de estudiantes, devolverá su nota, si no, devolverá *None*.

► **308** Tenemos los tiempos de cada ciclista y etapa participantes en la última vuelta ciclista local. La lista *ciclistas* contiene una serie de nombres. La matriz *tiempos* tiene una fila por cada ciclista, en el mismo orden con que aparecen en *ciclistas*. Cada fila tiene el tiempo en segundos (un valor flotante) invertido en cada una de las 5 etapas de la carrera. ¿Complicado? Este ejemplo te ayudará: te mostramos a continuación un ejemplo de lista *ciclistas* y de matriz *tiempos* para 3 corredores.

```

1 ciclistas = ['Pere_Porcar', 'Joan_Beltran', 'Lledó_Fabra']
2 tiempo = [[10092.0, 12473.1, 13732.3, 10232.1, 10332.3],
3           [11726.2, 11161.2, 12272.1, 11292.0, 12534.0],
4           [10193.4, 10292.1, 11712.9, 10133.4, 11632.0]]

```

En el ejemplo, el ciclista Joan Beltran invirtió 11161.2 segundos en la segunda etapa.

Se pide:

- Una función que reciba la lista y la matriz y devuelva el ganador de la vuelta (aquel cuya suma de tiempos en las 5 etapas es mínima).
- Una función que reciba la lista, la matriz y un número de etapa y devuelva el nombre del ganador de la etapa.
- Un procedimiento que reciba la lista, la matriz y muestre por pantalla el ganador de cada una de las etapas.

► **309** ¿Qué aparecerá por pantalla al ejecutar este programa?

```

1 a = 1
2 b = 2
3 [a, b] = [b, a]
4 print a, b

```

► **310** Diseña una función que reciba una lista de enteros y devuelva los números mínimo y máximo de la lista simultáneamente.

► **311** Diseña una función que reciba los tres coeficientes de una ecuación de segundo grado de la forma $ax^2 + bx + c = 0$ y devuelva una lista con sus soluciones reales. Si la ecuación sólo tiene una solución real, devuelve una lista con dos copias de la misma. Si no tiene solución real alguna o si tiene infinitas soluciones devuelve una lista con dos copias del valor *None*.

► **312** Diseña una función que reciba una lista de palabras (cadenas) y devuelva, simultáneamente, la primera y la última palabras según el orden alfabético.

► **313** Modifica Memori3 para que se ofrezca al usuario jugar con tres niveles de dificultad:

- Fácil: tablero de 3×4 .
- Normal: tablero de 4×6 .
- Difícil: tablero de 6×8 .

► **314** Implementa Memori3, una variante de Memori3 en el que hay que emparejar grupos de 3 letras iguales. (Asegúrate de que el número de casillas de la matriz sea múltiplo de 3.)

► **315** Construye el programa del Buscaminas inspirándote en la forma en que hemos desarrollado el juego Memori3. Te damos unas pistas para ayudarte en la implementación:

- Crea una matriz cuyas casillas contengan el valor *True* o *False*. El primer valor indica que hay una mina en esa casilla. Ubica las minas al azar. El número de minas dependerá de la dificultad del juego.
- Crea una matriz que contenga el número de minas que rodean a cada casilla. Calcula esos valores a partir de la matriz de minas. Ojo con las casillas «especiales»: el número de vecinos de las casillas de los bordes requiere un cuidado especial.
- Dibuja las minas y baldosas que las tapan. Define adecuadamente el sistema de coordenadas del lienzo.
- Usa una rutina de control del ratón similar a la desarrollada para Memori3. Te interesa detectar dos pulsaciones de ratón distintas: la del botón 1, que asociamos a «descubre casilla», y la del botón 3, que asociamos a «marcar posición». La marca de posición es una señal que dispone el usuario en una casilla para indicar que él cree que oculta una mina. Necesitarás una nueva matriz de marcas.
- El programa principal es un bucle similar al de Memori3. El bucle principal finaliza cuando hay una coincidencia total entre la matriz de bombas y la matriz de marcas puestas por el usuario.
- Cada vez que se pulse el botón 1, destruye la baldosa correspondiente. Si ésta escondía una mina, la partida ha acabado y el jugador ha muerto. Si no, crea un objeto gráfico (texto) que muestre el número de minas vecinas a esa casilla.
- Cada vez que se pulse el botón 3, añade una marca a la casilla correspondiente si no la había, y elimina la que había en caso contrario.

► **316** Modifica el Buscaminas para que cada vez que se pulse con el primer botón en una casilla con cero bombas vecinas, se marquen todas las casillas alcanzables desde esta y que no tienen bomba. (Este ejercicio es difícil. Piensa bien en la estrategia que has de seguir.)

- **465** Haz un programa que lea un fichero de texto que puede contener vocales acentuadas y muestre por pantalla una versión del mismo en el que cada vocal acentuada ha sido sustituida por la misma vocal sin acentuar.
- **466** Diseña un programa, `descifra.py`, que descifre ficheros cifrados por `cifra.py`. El programa pedirá el nombre del fichero cifrado y el del fichero en el que se guardará el resultado.
- **467** Diseña un programa que, dados dos ficheros de texto, nos diga si el primero es una versión cifrada del segundo (con el código de cifrado descrito en la sección).
- **468** Diseña un programa que obtenga los 100 primeros números primos y los almacene en un fichero de texto llamado `primos.txt`.
- **469** Haz un programa que pida el nombre de un grupo de usuarios Unix. A continuación, abre en modo escritura un fichero con el mismo nombre del grupo leído y extensión `grp`. En dicho fichero deberás escribir el nombre real de todos los usuarios de dicho grupo, uno en cada línea. (Lee antes el enunciado de los ejercicios [461](#) y [463](#).)
- **470** Deseamos automatizar el envío personalizado de correo electrónico a nuestros clientes. (¿Recuerdas el apartado ??? Si no, estúdialo de nuevo.) Disponemos de un fichero de clientes llamado `clientes.txt` en el que cada línea tiene la dirección de correo electrónico y el nombre de un cliente nuestro. El fichero empieza así:

```
1 al00000@alumail.uji.es_Pedro_Pérez
2 spammer@spam.com_John_Doe
3 ...
```

En otro fichero, llamado `carta.txt`, tenemos un carta personalizable. En ella, el lugar donde queremos poner el nombre del cliente aparece marcado con el texto `$CLIENTE$`. La carta empieza así:

```
1 Estimado/a_Sr/a_$CLIENTE$:
2
3 Tenemos_noticias_de_que_ud.,_don/doña_$CLIENTE$,_no_ha_abonado_el_importe
4 de_la_cuota_mensual_a_que_le_obliga_el_draconiano_contrato_que_firmó
5 ...
```

Haz un programa que envíe un correo a cada cliente con el contenido de `carta.txt` debidamente personalizado. Ahora que sabes definir y usar funciones, diseña el programa sirviéndote de ellas.

- **471** Nuestro ficheros `clientes.txt` se modifica ahora para incluir como segundo campo de cada línea el sexo de la persona. La letra H indica que se trata de un hombre y la letra M que se trata de una mujer. Modifica el programa para que sustituya las expresiones `don/doña` por `don` o `doña`, `Estimado/a` por `Estimado` o `Estimada` y `Sr/a` por `Sr` o `Sra` según convenga.
- **472** Hemos decidido sustituir las tres llamadas al método `write` de las líneas 32, 33 y 34 por una sola:

```
fcopia.write(linea1+linea2+linea3)
```

¿Funcionará igual?

- **473** En su versión actual, es posible añadir dos veces una misma entrada a la agenda. Modifica `anyadir_entrada` para que sólo añada una nueva entrada si corresponde a una persona diferente. Añadir por segunda vez los datos de una misma persona supone sustituir el viejo teléfono por el nuevo.
- **474** Añade a la agenda las siguientes operaciones:
 - Listado completo de la agenda por pantalla. Cada entrada debe ocupar una sólo línea en pantalla.
 - Listado de teléfonos de todas las personas cuyo apellido empieza por una letra determinada.
- **475** Haz que cada vez que se añade una entrada a la agenda, ésta quede ordenada alfabéticamente.
- **476** Deseamos poder trabajar con más de un teléfono por persona. Modifica el programa de la agenda para que la línea que contiene el teléfono contenga una relación de teléfonos separados por blancos. He aquí un ejemplo de entrada con tres teléfonos:

```
1 Pedro ↵
2 López ↵
3 964112537_964009923_96411092 ↵
```

La función `buscar_entrada` devolverá una lista con tantos elementos como teléfonos tiene la persona encontrada. Enriquece la aplicación con la posibilidad de borrar uno de los teléfonos de una persona.