

► **1** ¿Cuál es el máximo valor que puede representarse con 16 bits y un sistema de representación posicional como el descrito? ¿Qué secuencia de bits le corresponde?

► **2** ¿Cuántos bits se necesitan para representar los números del 0 al 18, ambos inclusive?

► **3** Calcula las siguientes sumas de números codificados con 8 bits en el sistema posicional:

a) $01111111 + 00000001$

b) $01010101 + 10101010$

c) $00000011 + 00000001$

► **4** Codifica en complemento a dos de 8 bits los siguientes valores:

a) 4

b) -4

c) 0

d) 127

e) 1

f) -1

► **5** Efectúa las siguientes sumas y restas en complemento a dos de 8 bits:

a) $4 + 4$

b) $-4 + 3$

c) $127 - 128$

d) $128 - 127$

e) $1 - 1$

f) $1 - 2$

► **6** Ejecuta paso a paso el mismo programa con los valores 2, -2 y 0 en las posiciones de memoria 10, 11 y 12, respectivamente.

► **7** Diseña un programa que calcule la media de cinco números depositados en las posiciones de memoria que van de la 10 a la 14 y que deje el resultado en la dirección de memoria 15. Recuerda que la media \bar{x} de cinco números x_1, x_2, x_3, x_4 y x_5 es

$$\bar{x} = \frac{\sum_{i=1}^5 x_i}{5} = \frac{x_1 + x_2 + x_3 + x_4 + x_5}{5}.$$

► **8** Diseña un programa que calcule la varianza de cinco números depositados en las posiciones de memoria que van de la 10 a la 14 y que deje el resultado en la dirección de memoria 15. La varianza, que se denota con σ^2 , es

$$\sigma^2 = \frac{\sum_{i=1}^5 (x_i - \bar{x})^2}{5},$$

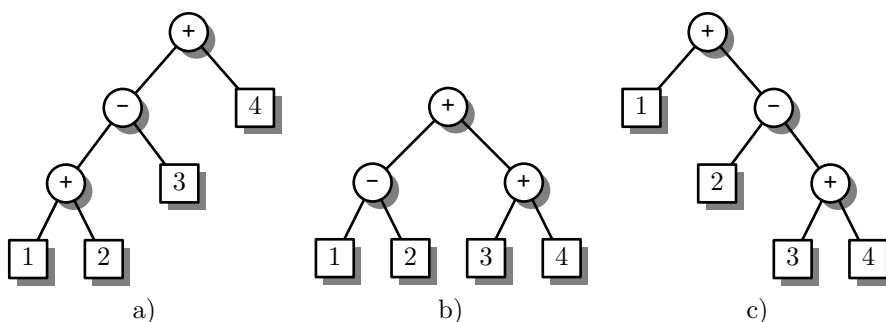
donde \bar{x} es la media de los cinco valores. Supón que existe una instrucción «Multiplicar el contenido de dirección a por el contenido de dirección b y dejar el resultado en dirección c ».

► **9** Diseña un algoritmo para calcular el área de un círculo dado su radio. (Recuerda que el área de un círculo es π veces el cuadrado del radio.)

► **10** Diseña un algoritmo que calcule el IVA (16%) de un producto dado su precio de venta sin IVA.

► **11** ¿Podemos llamar algoritmo a un procedimiento que escriba en una cinta de papel *todos* los números decimales de π ?

► **12** ¿Qué expresiones Python permiten, utilizando el menor número posible de paréntesis, efectuar *en el mismo orden* los cálculos representados con estos árboles sintácticos?



► **13** Dibuja los árboles sintácticos correspondientes a las siguientes expresiones aritméticas:

a) $1 + 2 + 3 + 4$

b) $1 - 2 - 3 - 4$

c) $1 - (2 - (3 - 4) + 1)$

► **14** ¿Qué resultados se obtendrán al evaluar las siguientes expresiones Python? Dibuja el árbol sintáctico de cada una de ellas, calcula a mano el valor resultante de cada expresión y comprueba, con la ayuda del ordenador, si tu resultado es correcto.

- a) $2 + 3 + 1 + 2$ c) $(2 + 3) * 1 + 2$ e) $+-+--6$
 b) $2 + 3 * 1 + 2$ d) $(2 + 3) * (1 + 2)$ f) $-++-+6$

► **15** Traduce las siguientes expresiones matemáticas a Python y evalúalas. Trata de utilizar el menor número de paréntesis posible.

- a) $2 + (3 \cdot (6/2))$ c) $(4/2)^5$ e) $(-3)^2$
 b) $\frac{4+6}{2+3}$ d) $(4/2)^{5+1}$ f) $-(3^2)$

(Nota: El resultado de evaluar cada expresión es: a) 11; b) 2; c) 32; d) 64; e) 9; f) -9 .)

► **16** ¿Qué resultará de evaluar las siguientes expresiones? Presta especial atención al tipo de datos que resulta de cada operación individual. Haz los cálculos a mano ayudándote con árboles sintácticos y comprueba el resultado con el ordenador.

- a) $1 / 2 / 4.0$ g) $4.0 ** (1 / 2) + 1 / 2$
 b) $1 / 2.0 / 4.0$ h) $4.0 ** (1.0 / 2) + 1 / 2.0$
 c) $1 / 2.0 / 4$ i) $3e3 / 10$
 d) $1.0 / 2 / 4$ j) $10 / 5e-3$
 e) $4 ** .5$ k) $10 / 5e-3 + 1$
 f) $4.0 ** (1 / 2)$ l) $3 / 2 + 1$

► **17** ¿Qué resultados se muestran al evaluar estas expresiones?

```
>>> True == True != False ↵
>>> 1 < 2 < 3 < 4 < 5 ↵
>>> (1 < 2 < 3) and (4 < 5) ↵
>>> 1 < 2 < 4 < 3 < 5 ↵
>>> (1 < 2 < 4) and (3 < 5) ↵
```

► **18** ¿Son válidos los siguientes identificadores?

- a) *Identificador* g) *desviación* m) *UnaVariable* r) *área*
 b) *Indice\dos* h) *año* n) *a(b)* s) *area-rect*
 c) *Dos palabras* i) *from* ñ) *12* t) *x_____1*
 d) *--* j) *var!* o) *uno.dos* u) *_____1*
 e) *12horas* k) *'var'* p) *x* v) *_x_*
 f) *hora12* l) *import_from* q) *π* w) *x_x*

► **19** ¿Qué resulta de ejecutar estas tres líneas?

```
>>> x = 10 ↵
>>> x = x * 10 ↵
>>> x ↵
```

► **20** Evalúa el polinomio $x^4 + x^3 + 2x^2 - x$ en $x = 1.1$. Utiliza variables para evitar teclear varias veces el valor de x . (El resultado es 4.1151.)

► **21** Evalúa el polinomio $x^4 + x^3 + \frac{1}{2}x^2 - x$ en $x = 10$. Asegúrate de que el resultado sea un número flotante. (El resultado es 11040.0.)

► **22** ¿Qué resultará de ejecutar las siguientes sentencias?

```
>>> z = 2 ↵
>>> z += 2 ↵
>>> z += 2 - 2 ↵
>>> z *= 2 ↵
```

```
>>> z *= 1 + 1 ↵
>>> z /= 2 ↵
>>> z %= 3 ↵
>>> z /= 3 - 1 ↵
>>> z -= 2 + 1 ↵
>>> z -= 2 ↵
>>> z **= 3 ↵
>>> z ↵
```

► **23** Evalúa estas expresiones y sentencias en el orden indicado:

- a) $a = 'b'$
- b) $a + 'b'$
- c) $a + 'a'$
- d) $a * 2 + 'b' * 3$
- e) $2 * (a + 'b')$

► **24** ¿Qué resultados se obtendrán al evaluar las siguientes expresiones y asignaciones Python? Calcula primero a mano el valor resultante de cada expresión y comprueba, con la ayuda del ordenador, si tu resultado es correcto.

- a) $'a' * 3 + '/' * 5 + 2 * 'abc' + '+'$
- b) $palindromo = 'abcba'$
 $(4 * '<' + palindromo + '>' * 4) * 2$
- c) $subcadena = '=' + '-' * 3 + '='$
 $'10' * 5 + 4 * subcadena$
- d) $2 * '12' + '.' + '3' * 3 + 'e-' + 4 * '76'$

► **25** Identifica regularidades en las siguientes cadenas, y escribe expresiones que, partiendo de subcadenas más cortas y utilizando los operadores de concatenación y repetición, produzcan las cadenas que se muestran. Introduce variables para formar las expresiones cuando lo consideres oportuno.

- a) $'%%%%%%%%././.<-><->'$
- b) $'(@)(@)(@)=====(@)(@)(@)====='$
- c) $'asdfasdfasdf-----??????asdfasdf'$
- d) $'.....*****-----*****-----.....*****-----*****-----'$

► **26** ¿Qué resultados se muestran al evaluar estas expresiones?

```
>>> 'abalorio' < 'abecedario' ↵
>>> 'abecedario' < 'abecedario' ↵
>>> 'abecedario' <= 'abecedario' ↵
>>> 'Abecedario' < 'abecedario' ↵
>>> 'Abecedario' == 'abecedario' ↵
>>> 124 < 13 ↵
>>> '124' < '13' ↵
>>> 'a' < 'a' ↵
```

► **27** Calcula con una única expresión el valor absoluto del redondeo de -3.2 . (El resultado es 3.0 .)

► **28** Convierte (en una única expresión) a una cadena el resultado de la división $5011/10000$ redondeado con 3 decimales.

► **29** ¿Qué resulta de evaluar estas expresiones?

```
>>> str(2.1) + str(1.2) ↵
>>> int(str(2) + str(3)) ↵
>>> str(int(12.3)) + '0' ↵
>>> int('2'+'3') ↵
>>> str(2 + 3) ↵
>>> str(int(2.1) + float(3)) ↵
```

► **30** ¿Qué resultados se obtendrán al evaluar las siguientes expresiones Python? Calcula primero a mano el valor resultante de cada expresión y comprueba, con la ayuda del ordenador, si tu resultado es correcto.

a) `int(exp(2 * log(3)))`

b) `round(4 * sin(3 * pi / 2))`

c) `abs(log10(.01) * sqrt(25))`

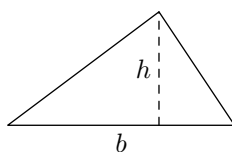
d) `round(3.21123 * log10(1000), 3)`

► **31** Diseña un programa que, a partir del valor del lado de un cuadrado (3 metros), muestre el valor de su perímetro (en metros) y el de su área (en metros cuadrados).

(El perímetro debe darte 12 metros y el área 9 metros cuadrados.)

► **32** Diseña un programa que, a partir del valor de la base y de la altura de un triángulo (3 y 5 metros, respectivamente), muestre el valor de su área (en metros cuadrados).

Recuerda que el área A de un triángulo se puede calcular a partir de la base b y la altura h como $A = \frac{1}{2}bh$.



(El resultado es 7.5 metros cuadrados.)

► **33** Diseña un programa que, a partir del valor de los dos lados de un rectángulo (4 y 6 metros, respectivamente), muestre el valor de su perímetro (en metros) y el de su área (en metros cuadrados).

(El perímetro debe darte 20 metros y el área 24 metros cuadrados.)

► **34** Diseña un programa que pida el valor del lado de un cuadrado y muestre el valor de su perímetro y el de su área.

(Prueba que tu programa funciona correctamente con este ejemplo: si el lado vale 1.1, el perímetro será 4.4, y el área 1.21.)

► **35** Diseña un programa que pida el valor de los dos lados de un rectángulo y muestre el valor de su perímetro y el de su área.

(Prueba que tu programa funciona correctamente con este ejemplo: si un lado mide 1 y el otro 5, el perímetro será 12.0, y el área 5.0.)

► **36** Diseña un programa que pida el valor de la base y la altura de un triángulo y muestre el valor de su área.

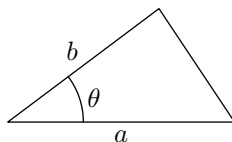
(Prueba que tu programa funciona correctamente con este ejemplo: si la base es 10 y la altura 100, el área será 500.0.)

► **37** Diseña un programa que pida el valor de los tres lados de un triángulo y calcule el valor de su área y perímetro.

Recuerda que el área A de un triángulo puede calcularse a partir de sus tres lados, a , b y c , así: $A = \sqrt{s(s-a)(s-b)(s-c)}$, donde $s = (a + b + c)/2$.

(Prueba que tu programa funciona correctamente con este ejemplo: si los lados miden 3, 5 y 7, el perímetro será 15.0 y el área 6.49519052838.)

► **38** El área A de un triángulo se puede calcular a partir del valor de dos de sus lados, a y b , y del ángulo θ que éstos forman entre sí con la fórmula $A = \frac{1}{2}ab \sin(\theta)$. Diseña un programa que pida al usuario el valor de los dos lados (en metros), el ángulo que estos forman (en grados), y muestre el valor del área.



(Ten en cuenta que la función `sin` de Python trabaja en radianes, así que el ángulo que leas en grados deberás pasarlo a radianes sabiendo que π radianes son 180 grados. Prueba que has hecho bien el programa introduciendo los siguientes datos: $a = 1$, $b = 2$, $\theta = 30$; el resultado es 0.5.)

► **39** Haz un programa que pida al usuario una cantidad de euros, una tasa de interés y un número de años. Muestra por pantalla en cuánto se habrá convertido el capital inicial transcurridos esos años si cada año se aplica la tasa de interés introducida.

Recuerda que un capital de C euros a un interés del x por cien durante n años se convierten en $C \cdot (1 + x/100)^n$ euros.

(Prueba tu programa sabiendo que una cantidad de 10 000 € al 4.5% de interés anual se convierte en 24 117.14 € al cabo de 20 años.)

► **40** Haz un programa que pida el nombre de una persona y lo muestre en pantalla repetido 1000 veces, pero dejando un espacio de separación entre aparición y aparición del nombre. (Utiliza los operadores de concatenación y repetición.)

► **41** ¿Qué mostrará por pantalla este programa?

```
1 print '%d' % 1
2 print '%d_%d' % (1, 2)
3 print '%d%d' % (1, 2)
4 print '%d,_%d' % (1, 2)
5 print 1, 2
6 print '%d_2' % 1
```

► **42** Un alumno inquieto ha experimentado con las marcas de formato y el método *upper* y ha obtenido un resultado sorprendente:

```
>>> print ('número_{}_d_y_número_{}_d' % (1, 2)).upper() ↵
NÚMERO 1 Y NÚMERO 2
>>> print 'número_{}_d_y_número_{}_d'.upper() % (1, 2) ↵
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ValueError: unsupported format character 'D' (0x44) at index 8
```

¿Qué crees que ha pasado?

(Nota: Aunque experimentar conlleva el riesgo de equivocarse, no podemos enfatizar suficientemente cuán importante es para que asimiles las explicaciones. Probarlo todo, cometer errores, reflexionar sobre ellos y corregirlos es uno de los mejores ejercicios imaginables.)

► **43** ¿Qué pequeña diferencia hay entre el programa `saluda.py` y este otro cuando los ejecutamos?

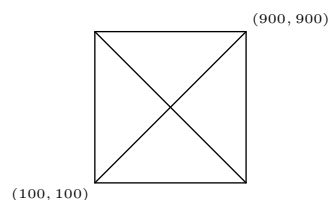
`saluda2.py`

```
1 nombre = raw_input('Tu nombre: ')
2 print 'Hola,', nombre, '.'
```

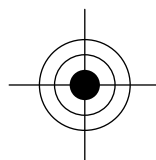
► **44** La marca `%s` puede representar cadenas con un número fijo de casillas. A la vista de cómo se podía expresar esta característica en la marca de enteros `%d`, ¿sabrías como indicar que deseamos representar una cadena que ocupa 10 casillas?

► **45** Diseña un programa que solicite el radio de una circunferencia y muestre su área y perímetro con sólo 2 decimales.

► **46** Dibuja esta figura. (Te indicamos las coordenadas de las esquinas inferior izquierda y superior derecha.)

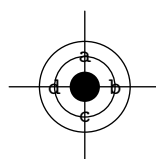


► **47** Dibuja esta figura.



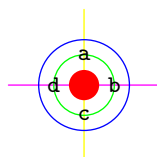
Los tres círculos concéntricos tienen radios 100, 200 y 300, respectivamente.

► **48** Dibuja esta figura.



Los tres círculos concéntricos tienen radios 100, 200 y 300, respectivamente.

► 49 Dibuja esta figura.

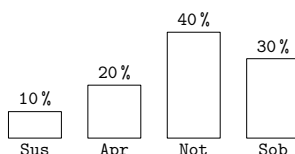


(Hemos usado los colores amarillo y magenta para las líneas rectas, verde y azul para los círculos y negro para las letras.)

► 50 Modifica el programa para que sea el usuario quien proporcione, mediante el teclado, el valor del porcentaje de suspensos, aprobados, notables y sobresalientes.

► 51 Modifica el programa para que sea el usuario quien proporcione, mediante el teclado, el *número* de suspensos, aprobados, notables y sobresalientes. (Antes de dibujar el gráfico de pastel debes convertir esas cantidades en porcentajes.)

► 52 Queremos representar la información de forma diferente: mediante un gráfico de barras. He aquí cómo:



Diseña un programa que solicite por teclado el número de personas con cada una de las cuatro calificaciones y muestre el resultado con un gráfico de barras.

► 53 Un programador propone el siguiente programa para resolver la ecuación de primer grado:

```
1 a = float(raw_input('Valor de a:'))
2 b = float(raw_input('Valor de b:'))
3
4 a * x + b = 0
5
6 print 'Solución:', x
```

¿Es correcto este programa? Si no, explica qué está mal.

► 54 Otro programador propone este programa:

```
1 x = -b / a
2
3 a = float(raw_input('Valor de a:'))
4 b = float(raw_input('Valor de b:'))
5
6 print 'Solución:', x
```

¿Es correcto? Si no lo es, explica qué está mal.

► 55 Un estudiante ha tecleado el último programa y, al ejecutarlo, obtiene este mensaje de error.

```
File "primer_grado4.py", line 7
    if a = 0:
        ^
SyntaxError: invalid syntax
```

Aquí tienes el contenido del fichero que él ha escrito:

```
primer_grado.3.py
1 a = float(raw_input('Valor de a:'))
2 b = float(raw_input('Valor de b:'))
3
4 if a != 0:
5     x = -b/a
6     print 'Solución:', x
7 if a = 0:
8     print 'La ecuación no tiene solución.'
```

Por más que el estudiante lee el programa, no encuentra fallo alguno. Él dice que la línea 7, que es la marcada como errónea, se lee así: «si *a* es igual a cero...» ¿Está en lo cierto? ¿Por qué se detecta un error?

► **56** Un programador primerizo cree que la línea 7 de la última versión de `primer_grado.py` es innecesaria, así que propone esta otra versión como solución válida:

```

primer_grado.4.py
1 a = float(raw_input('Valor de a:'))
2 b = float(raw_input('Valor de b:'))
3
4 if a != 0:
5     x = -b/a
6     print 'Solución:', x
7
8 print 'La ecuación no tiene solución.'
```

Haz una traza del programa para $a = 2$ y $b = 2$. ¿Son correctos todos los mensajes que muestra por pantalla el programa?

► **57** Indica qué líneas del último programa (y en qué orden) se ejecutarán para cada uno de los siguientes casos:

a) $a = 2$ y $b = 6$. b) $a = 0$ y $b = 3$. c) $a = 0$ y $b = -3$. d) $a = 0$ y $b = 0$.

► **58** Diseña un programa que lea un número flotante por teclado y muestre por pantalla el mensaje «El número es negativo.» sólo si el número es menor que cero.

► **59** Diseña un programa que lea un número flotante por teclado y muestre por pantalla el mensaje «El número es positivo.» sólo si el número es mayor o *igual* que cero.

► **60** Diseña un programa que lea la edad de dos personas y diga quién es más joven, la primera o la segunda. Ten en cuenta que ambas pueden tener la misma edad. En tal caso, hazlo saber con un mensaje adecuado.

► **61** Diseña un programa que lea un carácter de teclado y muestre por pantalla el mensaje «Es paréntesis» sólo si el carácter leído es un paréntesis abierto o cerrado.

► **62** Indica en cada uno de los siguientes programas qué valores en las respectivas entradas provocan la aparición de los distintos mensajes. Piensa primero la solución y comprueba luego que es correcta ayudándote con el ordenador.

a) `misterio.3.py` `misterio.py`

```

1 letra = raw_input('Dame una letra minúscula:')
2
3 if letra <= 'k':
4     print 'Es de las primeras del alfabeto'
5 if letra >= 'l':
6     print 'Es de las últimas del alfabeto'
```

b) `misterio.4.py` `misterio.py`

```

1 from math import ceil # ceil redondea al alza.
2
3 grados = float(raw_input('Dame un ángulo (en grados):'))
4
5 cuadrante = int(ceil(grados) % 360) / 90
6 if cuadrante == 0:
7     print 'primer cuadrante'
8 if cuadrante == 1:
9     print 'segundo cuadrante'
10 if cuadrante == 2:
11     print 'tercer cuadrante'
12 if cuadrante == 3:
13     print 'cuarto cuadrante'
```

► **63** ¿Qué mostrará por pantalla el siguiente programa?

```

comparaciones.py
1 if 14 < 120:
2     print 'Primer saludo'
3 if '14' < '120':
4     print 'Segundo saludo'
```

► **64** Diseña un programa que, dado un número entero, muestre por pantalla el mensaje «El número es par.» cuando el número sea par y el mensaje «El número es impar.» cuando sea impar.

(Una pista: un número es par si el resto de dividirlo por 2 es 0, e impar en caso contrario.)

- **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
```



```

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)

```

- **71** Ejecuta paso a paso, con ayuda del entorno de depuración de PythonG, el programa del ejercicio anterior.
- **72** Diseña un programa Python que lea un carácter cualquiera desde el teclado, y muestre el mensaje «Es una MAYÚSCULA» cuando el carácter sea una letra mayúscula y el mensaje «Es una MINÚSCULA» cuando sea una minúscula. En cualquier otro caso, no mostrará mensaje alguno. (Considera únicamente letras del alfabeto inglés.) Pista: aunque parezca una obviedad, recuerda que una letra es minúscula si está entre la 'a' y la 'z', y mayúscula si está entre la 'A' y la 'Z'.
- **73** Amplía la solución al ejercicio anterior para que cuando el carácter introducido no sea una letra muestre el mensaje «No es una letra». (Nota: no te preocupes por las letras ñe, ce cedilla, vocales acentuadas, etc.)
- **74** Amplía el programa del ejercicio anterior para que pueda identificar las letras ñe minúscula y mayúscula.
- **75** Modifica el programa que propusiste como solución al ejercicio 66 sustituyendo todas las condiciones que sea posible por cláusulas **else** de condiciones anteriores.
- **76** Un programador ha intentado solucionar el problema del discriminante negativo con un programa que empieza así:

```

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 sqrt(b**2 - 4*a*c) >= 0:
9         x1 = (-b + sqrt(b**2 - 4*a*c)) / (2 * a)
10        x2 = (-b - sqrt(b**2 - 4*a*c)) / (2 * a)
11    ...

```

Evidentemente, el programa es incorrecto y te sorprenderá saber que algunos estudiantes proponen soluciones similares a ésta. El problema estriba en el posible valor negativo *del argumento* de *sqrt*, así que la comparación es incorrecta, pues pregunta por el signo *de la raíz* de dicho argumento. Pero el programa no llega siquiera a dar solución alguna (bien o mal calculada) cuando lo ejecutamos con, por ejemplo, $a = 4$, $b = 2$ y $c = 4$. ¿Qué sale por pantalla en ese caso? ¿Por qué?

- **77** ¿Qué líneas del último programa se ejecutan y qué resultado aparece por pantalla en cada uno de estos casos?

a) $a = 2$ y $b = 3$. b) $a = 3$ y $b = 2$. c) $a = -2$ y $b = 0$. d) $a = 1$ y $b = 1$.

Analiza con cuidado el último caso. Observa que los dos números son iguales. ¿Cuál es, pues, el máximo? ¿Es correcto el resultado del programa?

- **78** Un aprendiz de programador ha diseñado este otro programa para calcular el máximo de dos números:

```

maximo.2.py
1 a = int(raw_input('Dame el primer número: '))
2 b = int(raw_input('Dame el segundo número: '))
3
4 if a > b:
5     maximo = a
6 if b > a:
7     maximo = b
8
9 print 'El máximo es', maximo

```

¿Es correcto? ¿Qué pasa si introducimos dos números iguales?

- **79** ¿Qué secuencia de líneas de este último programa se ejecutará en cada uno de estos casos?

a) $a = 2$, $b = 3$ y $c = 4$. b) $a = 3$, $b = 2$ y $c = 4$. c) $a = 1$, $b = 1$ y $c = 1$.


Ayúdate con el modo de depuración de PythonG.

- **80** Diseña un programa que calcule el máximo de 5 números enteros. Si sigues una estrategia similar a la de la primera solución propuesta para el problema del máximo de 3 números, tendrás problemas. Intenta resolverlo como en el último programa de ejemplo, es decir con un «candidato a valor máximo» que se va actualizando al compararse con cada número.


- **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

```

► **86** La fórmula $C' = C \cdot (1 + x/100)^n$ nos permite obtener el capital final que lograremos a partir de un capital inicial (C), una tasa de interés anual (x) en tanto por cien y un número de años (n). Si lo que nos interesa conocer es el número de años n que tardaremos en lograr un capital final C' partiendo de un capital inicial C a una tasa de interés anual x , podemos despejar n en la fórmula del ejercicio **67** de la siguiente manera:

$$n = \frac{\log(C') - \log(C)}{\log(1 + x/100)}$$

Diseña un programa Python que obtenga el número de años que se tarda en conseguir un capital final dado a partir de un capital inicial y una tasa de interés anual también dados. El programa debe tener en cuenta cuándo se puede realizar el cálculo y cuándo no en función del valor de la tasa de interés (para evitar una división por cero, el cálculo de logaritmos de valores negativos, etc)... con una excepción: si C y C' son iguales, el número de años es 0 independientemente de la tasa de interés (incluso de la que provocaría un error de división por cero).

(Ejemplos: Para obtener 11 000 € por una inversión de 10 000 € al 5% anual es necesario esperar 1.9535 años. Obtener 11 000 € por una inversión de 10 000 € al 0% anual es imposible. Para obtener 10 000 € con una inversión de 10 000 € no hay que esperar nada, sea cual sea el interés.)

► **87** Diseña un programa que, dado un número real que debe representar la calificación numérica de un examen, proporcione la calificación cualitativa correspondiente al número dado. La calificación cualitativa será una de las siguientes: «Suspendido» (nota menor que 5), «Aprobado» (nota mayor o igual que 5, pero menor que 7), «Notable» (nota mayor o igual que 7, pero menor que 8.5), «Sobresaliente» (nota mayor o igual que 8.5, pero menor que 10), «Matrícula de Honor» (nota 10).

► **88** Diseña un programa que, dado un carácter cualquiera, lo identifique como vocal minúscula, vocal mayúscula, consonante minúscula, consonante mayúscula u otro tipo de carácter.

► **89** ¿Por qué obtenemos un error en esta sesión de trabajo con el intérprete interactivo?

```
>>> a = 0
>>> if 1/a > 1 and a != 0:
...     print a
...
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ZeroDivisionError: integer division or modulo by zero
```

► **90** Nuestro aprendiz de programador ha tecleado en su ordenador el último programa, pero se ha despistado y ha escrito esto:

```
circulo.3.py      circulo.py
1 from math import pi
2
3 radio = float(raw_input('Dame el radio de un círculo:'))
4
5 print 'Escoge una opción:'
6 print 'a) Calcular el diámetro.'
7 print 'b) Calcular el perímetro.'
8 print 'c) Calcular el área.'
9 opcion = raw_input('Teclea a, b o c y pulsa el retorno de carro:')
10
11 if opcion == a:
12     diametro = 2 * radio
13     print 'El diámetro es', diametro
14 else:
15     if opcion == b:
16         perimetro = 2 * pi * radio
17         print 'El perímetro es', perimetro
18     else:
19         if opcion == c:
20             area = pi * radio ** 2
21             print 'El área es', area
```

Las líneas sombreadas son diferentes de sus equivalentes del programa original. ¿Funcionará el programa del aprendiz? Si no es así, ¿por qué motivo?

► **91** Haz una traza del programa suponiendo que el usuario teclea la letra d cuando se le solicita una opción. ¿Qué líneas del programa se ejecutan?

- **92** El programa presenta un punto débil: si el usuario escribe una letra mayúscula en lugar de minúscula, no se selecciona ninguna opción. Modifica el programa para que también acepte letras mayúsculas.
- **93** Modifica la solución del ejercicio **87** usando ahora la estructura **elif**. ¿No te parece más legible la nueva solución?
- **94** Ejecuta el último programa paso a paso con el entorno de depuración de PythonG.
- **95** Haz una traza de este programa:

```
ejercicio_bucle.9.py      ejercicio_bucle.py
1 i = 0
2 while i <= 3:
3     print i
4     i += 1
5 print 'Hecho'
```

- **96** Haz una traza de este programa:

```
ejercicio_bucle.10.py     ejercicio_bucle.py
1 i = 0
2 while i < 10:
3     print i
4     i += 2
5 print 'Hecho'
```

- **97** Haz una traza de este programa:

```
ejercicio_bucle.11.py     ejercicio_bucle.py
1 i = 3
2 while i < 10:
3     i += 2
4     print i
5 print 'Hecho'
```

- **98** Haz una traza de este programa:

```
ejercicio_bucle.12.py     ejercicio_bucle.py
1 i = 1
2 while i < 100:
3     i *= 2
4     print i
```

- **99** Haz una traza de este programa:

```
ejercicio_bucle.13.py     ejercicio_bucle.py
1 i = 10
2 while i < 2:
3     i *= 2
4     print i
```

- **100** Haz unas cuantas trazas de este programa para diferentes valores de *i*.

```
ejercicio_bucle.14.py     ejercicio_bucle.py
1 i = int(raw_input('Valor inicial: '))
2 while i < 10:
3     print i
4     i += 1
```

¿Qué ocurre si el valor de *i* es mayor o igual que 10? ¿Y si es negativo?

- **101** Haz unas cuantas trazas de este programa para diferentes valores de *i* y de *limite*.

```
ejercicio_bucle.15.py     ejercicio_bucle.py
1 i = int(raw_input('Valor inicial: '))
2 limite = int(raw_input('Límite: '))
3 while i < limite:
4     print i
5     i += 1
```

- **102** Haz unas cuantas trazas de este programa para diferentes valores de *i*, de *limite* y de *incremento*.

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.

► **114** ¿Es correcto este otro programa? ¿En qué se diferencia del anterior? ¿Cuál te parece mejor (si es que alguno de ellos te parece mejor)?

```

circulo.4.py
circulo.py
1 from math import pi
2
3 radio = float(raw_input('Dame el radio de un círculo:'))
4
5 opcion = ''
6 while opcion < 'a' or opcion > 'c':
7     print 'Escoge una opción:'
8     print 'a) Calcular el diámetro.'
9     print 'b) Calcular el perímetro.'
10    print 'c) Calcular el área.'
11    opcion = raw_input('Teclea a, b o c y pulsa el retorno de carro:')
12    if opcion < 'a' or opcion > 'c':
13        print 'Sólo hay tres opciones: a, b o c. Tú has tecleado', opcion
14
15 if opcion == 'a':
16     diametro = 2 * radio
17     print 'El diámetro es', diametro
18 elif opcion == 'b':
19     perimetro = 2 * pi * radio
20     print 'El perímetro es', perimetro
21 elif opcion == 'c':
22     area = pi * radio ** 2
23     print 'El área es', area

```

► **115** El programa anterior pide el valor del radio al principio y, después, permite seleccionar uno o más cálculos con ese valor del radio. Modifica el programa para que pida el valor del radio cada vez que se solicita efectuar un nuevo cálculo.

► **116** Un vector en un espacio tridimensional es una tripleta de valores reales (x, y, z) . Deseamos confeccionar un programa que permita operar con dos vectores. El usuario verá en pantalla un menú con las siguientes opciones:

- 1) Introducir el primer vector
- 2) Introducir el segundo vector
- 3) Calcular la suma
- 4) Calcular la diferencia
- 5) Calcular el producto escalar
- 6) Calcular el producto vectorial
- 7) Calcular el ángulo (en grados) entre ellos
- 8) Calcular la longitud
- 9) Finalizar

Puede que necesites que te refresquemos la memoria sobre los cálculos a realizar. Si es así, la tabla 1 te será de ayuda:

Operación	Cálculo
Suma: $(x_1, y_1, z_1) + (x_2, y_2, z_2)$	$(x_1 + x_2, y_1 + y_2, z_1 + z_2)$
Diferencia: $(x_1, y_1, z_1) - (x_2, y_2, z_2)$	$(x_1 - x_2, y_1 - y_2, z_1 - z_2)$
Producto escalar: $(x_1, y_1, z_1) \cdot (x_2, y_2, z_2)$	$x_1x_2 + y_1y_2 + z_1z_2$
Producto vectorial: $(x_1, y_1, z_1) \times (x_2, y_2, z_2)$	$(y_1z_2 - z_1y_2, z_1x_2 - x_1z_2, x_1y_2 - y_1x_2)$
Ángulo entre (x_1, y_1, z_1) y (x_2, y_2, z_2)	$\frac{180}{\pi} \cdot \arccos\left(\frac{x_1x_2 + y_1y_2 + z_1z_2}{\sqrt{x_1^2 + y_1^2 + z_1^2}\sqrt{x_2^2 + y_2^2 + z_2^2}}\right)$
Longitud de (x, y, z)	$\sqrt{x^2 + y^2 + z^2}$

Tabla 1: Recordatorio de operaciones básicas sobre vectores.

Tras la ejecución de cada una de las acciones del menú éste reaparecerá en pantalla, a menos que la opción escogida sea la número 9. Si el usuario escoge una opción diferente, el programa advertirá al usuario de su error y el menú reaparecerá.

Las opciones 4 y 6 del menú pueden proporcionar resultados distintos en función del orden de los operandos, así que, si se escoge cualquiera de ellas, deberá mostrarse un nuevo menú que permita seleccionar el orden de los operandos. Por ejemplo, la opción 4 mostrará el siguiente menú:

- 1) Primer vector menos segundo vector
- 2) Segundo vector menos primer vector

Nuevamente, si el usuario se equivoca, se le advertirá del error y se le permitirá corregirlo.

La opción 8 del menú principal conducirá también a un submenú para que el usuario decida sobre cuál de los dos vectores se aplica el cálculo de longitud.

Ten en cuenta que tu programa debe contemplar y controlar toda posible situación excepcional: divisiones por cero, raíces con argumento negativo, etcétera. (Nota: La función arcocoseno se encuentra disponible en el módulo *math* y su identificador es *acos*.)

► **117** Haz un programa que muestre la tabla de multiplicar de un número introducido por teclado por el usuario. Aquí tienes un ejemplo de cómo se debe comportar el programa:

```
Dame un número: 5
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
```

► **118** Realiza un programa que proporcione el desglose en billetes y monedas de una cantidad entera de euros. Recuerda que hay billetes de 500, 200, 100, 50, 20, 10 y 5 € y monedas de 2 y 1 €. Debes «recorrer» los valores de billete y moneda disponibles con uno o más bucles **for-in**.

► **119** Haz un programa que muestre la raíz n -ésima de un número leído por teclado, para n tomando valores entre 2 y 100.

► **120** Haz un programa que muestre, en líneas independientes, todos los números pares comprendidos entre 0 y 200 (ambos inclusive).

► **121** Haz un programa que muestre, en líneas independientes y en orden inverso, todos los números pares comprendidos entre 0 y 200 (ambos inclusive).

► **122** Escribe un programa que muestre los números pares positivos entre 2 y un número cualquiera que introduzca el usuario por teclado.

► **123** Haz un programa que pida el valor de dos enteros n y m y que muestre por pantalla el valor de

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

Debes usar un bucle **for-in** para el cálculo del sumatorio.

► **124** Haz un programa que pida el valor de dos enteros n y m y que muestre por pantalla el valor de

$$\sum_{i=n}^m i^2.$$

► **125** Haz un programa que pida el valor de dos enteros n y m y calcule el sumatorio de todos los números pares comprendidos entre ellos (incluyéndolos en el caso de que sean pares).

► **126** Haz una traza del programa para los siguientes números:

- a) 4 b) 13 c) 18 d) 2 (¡jojo con éste!)

► **127** Haz una traza del último programa para el número 125.


► **128** Haz una traza del último programa para el número 125.

► **129** Haz un programa que calcule el máximo común divisor (mcd) de dos enteros positivos. El mcd es el número más grande que divide exactamente a ambos números.


- **130** Haz un programa que calcule el máximo común divisor (mcd) de tres enteros positivos. El mcd de tres números es el número más grande que divide exactamente a los tres.
- **131** Haz una traza del programa para el valor 125.
- **132** En realidad no hace falta explorar todo el rango de números entre 2 y $n - 1$ para saber si un número n es o no es primo. Basta con explorar el rango de números entre 2 y la parte entera de $n/2$. Piensa por qué. Modifica el programa para que sólo exploremos ese rango.
- **133** Ni siquiera hace falta explorar todo el rango de números entre 2 y $n/2$ para saber si un número n es o no es primo. Basta con explorar el rango de números entre 2 y la parte entera de \sqrt{n} . (Créetelo.) Modifica el programa para que sólo exploremos ese rango.
- **134** Haz un programa que vaya leyendo números y mostrándolos por pantalla hasta que el usuario introduzca un número negativo. En ese momento, el programa mostrará un mensaje de despedida y finalizará su ejecución.
- **135** Haz un programa que vaya leyendo números hasta que el usuario introduzca un número negativo. En ese momento, el programa mostrará por pantalla el número mayor de cuantos ha visto.
- **136** ¿Qué resultará de ejecutar estos programas?

a)  `ejercicio_for.7.py` `ejercicio_for.py`

```
1 for i in range(0, 5):
2     for j in range(0, 3):
3         print i, j
```

b)  `ejercicio_for.8.py` `ejercicio_for.py`


```
1 for i in range(0, 5):
2     for j in range(i, 5):
3         print i, j
```

c)  `ejercicio_for.9.py` `ejercicio_for.py`


```
1 for i in range(0, 5):
2     for j in range(0, i):
3         print i, j
```

d)  `ejercicio_for.10.py` `ejercicio_for.py`

```
1 for i in range(0, 4):
2     for j in range(0, 4):
3         for k in range(0, 2):
4             print i, j, k
```

e)  `ejercicio_for.11.py` `ejercicio_for.py`

```
1 for i in range(0, 4):
2     for j in range(0, 4):
3         for k in range(i, j):
4             print i, j, k
```

f)  `ejercicio_for.12.py` `ejercicio_for.py`

```
1 for i in range(1, 5):
2     for j in range(0, 10, i):
3         print i, j
```

- **137** Haz un programa que muestre la función coseno en el intervalo que te indique el usuario.
- **138** Modifica el programa anterior para que se muestren dos funciones a la vez: la función seno y la función coseno, pero cada una en un color distinto.
- **139** Haz un programa que muestre la función $1/(x + 1)$ en el intervalo $[-2, 2]$ con 100 puntos azules. Ten en cuenta que la función es «problemática» en $x = -1$, por lo que dibujaremos un punto rojo en las coordenadas $(-1, 0)$.
- **140** Haz un programa que, dados tres valores a , b y c , muestre la función $f(x) = ax^2 + bx + c$ en el intervalo $[z_1, z_2]$, donde z_1 y z_2 son valores proporcionados por el usuario. El programa de dibujo debe calcular el valor máximo y mínimo de $f(x)$ en el intervalo indicado para ajustar el valor de `window_coordinates` de modo que la función se muestre sin recorte alguno.
- **141** Añade a la gráfica del ejercicio anterior una representación de los ejes coordenados en color azul. Dibuja con círculos rojos los puntos en los que la parábola $f(x)$ corta el eje horizontal. Recuerda que la parábola corta al eje horizontal en los puntos x_1 y x_2 que son solución de la ecuación de segundo grado $ax^2 + bx + c = 0$.

- **142** ¿Qué pasaría si los dos cuerpos ocuparan exactamente la misma posición en el plano? Modifica el programa para que, si se da el caso, no se produzca error alguno y finalice inmediatamente la ejecución del bucle.
- **143** Modifica el programa para que la simulación no finalice nunca (bueno, sólo cuando el usuario interrumpa la ejecución del programa).
- **144** ¿Serías capaz de extender el programa para que muestre la interacción entre tres cuerpos? Repasa la formulación física del problema antes de empezar a programar.
- **145** Modifica el juego para que la barra que indica el combustible disponible se ponga de color rojo cuando quede menos del 25%.
- **146** Modifica el juego para que el usuario pueda escoger, con un menú, un nivel de dificultad. Ofrece al menos tres niveles: fácil, normal y difícil. Puedes modificar la dificultad del juego a voluntad alterando parámetros como el fuel disponible, el consumo, la fuerza de la gravedad, la velocidad de desplazamiento de la plataforma, etc.
- **147** Modifica el juego para que la plataforma no esté en el suelo, sino flotando. El usuario debe aterrizar en la plataforma *desde arriba*, claro está. Si se golpea a la plataforma desde abajo, la nave se destruirá y el jugador habrá fracasado.
- **148** Añade efectos especiales al juego. Por ejemplo, cambia el color del fondo para que sea negro y añade unas estrellas. También puedes mostrar una líneas amarillas saliendo de la nave cuando se activa algún propulsor. Si se acciona el propulsor inferior, la líneas saldrán de debajo de la nave, y si se activa un propulsor lateral, las líneas saldrán del lado correspondiente.
- **149** Modifica el juego para que aparezca un número determinado de meteoritos en pantalla (tres, por ejemplo). Cada meteorito se representará con un círculo de color rojo y se irá desplazando por la pantalla. Si la nave toca un meteorito, ésta se destruirá.
- **150** Programa un juego de frontón electrónico. El usuario controlará una raqueta en el lado inferior de la pantalla. Con la raqueta podrá golpear una pelota que rebotará en las paredes. Si la pelota se sale por el borde inferior de la pantalla, el juego finaliza.
- **151** Modifica el juego del frontón para convertirlo en un teletenis. El ordenador controlará una raqueta en el lado superior de la imagen. No permitas que el ordenador haga trampas, es decir, la velocidad de desplazamiento de la raqueta ha de ser (como mucho) la misma que la del usuario.
- **152** ¿Qué se mostrará en pantalla al ejecutar estas sentencias?

```
>>> print '\\n' ↵
>>> print '\\157\\143\\164\\141\\154' ↵
>>> print '\\t\\tuna\\bo' ↵
```

(Te recomendamos que resuelvas este ejercicio a mano y compruebes la validez de tus respuestas con ayuda del ordenador.)

- **153** ¿Cómo crees que se pueden representar dos barras invertidas seguidas en una cadena?
- **154** La secuencia de escape `\a` emite un aviso sonoro (la «campana»). ¿Qué hace exactamente cuando se imprime en pantalla? Ejecuta `print '\a'` y lo averiguarás.
- **155** Averigua el código ASCII de los 10 primeros caracteres de la tabla ??.
- **156** La última letra del DNI puede calcularse a partir de sus números. Para ello sólo tienes que dividir el número por 23 y quedarte con el resto. El resto 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

Diseña un programa que lea de teclado un número de DNI y muestre en pantalla la letra que le corresponde.

(Nota: una implementación basada en tomar una decisión con **if-elif** conduce a un programa muy largo. Si usas el operador de indexación de cadenas de forma inteligente, el programa apenas ocupa tres líneas. Piensa cómo.)

- **157** Intentamos mostrar los caracteres de la cadena en orden inverso así:

```
>>> a = "mi_cadena" ↵
>>> for i in range(len(a), -1): ↵
...     print a[i] ↵
... ↵
```

¿Funciona?

- **158** Intentamos mostrar los caracteres de la cadena en orden inverso así:

```
>>> a = "mi_cadena"
>>> for i in range(len(a)-1, -1, -1):
...     print a[i]
... 
```

¿Funciona?

- **159** Diseña un programa que lea una cadena y muestre el número de espacios en blanco que contiene.
- **160** Diseña un programa que lea una cadena y muestre el número de letras mayúsculas que contiene.
- **161** Diseña una programa que lea una cadena y muestre en pantalla el mensaje «**Contiene dígito**» si contiene algún dígito y «**No contiene dígito**» en caso contrario.
- **162** Haz una traza del programa para la cadena 'a_b'. ¿Qué líneas se ejecutan y qué valores toman las variables *cambios*, *anterior* y *caracter* tras la ejecución de cada una de ellas?
- **163** Ídem para la cadena 'a_b'.
- **164** ¿Funciona el programa cuando introducimos una cadena formada sólo por espacios en blanco? ¿Por qué? Si su comportamiento no te parece normal, corrígelo.
- **165** Modifica el programa para que base el cómputo de palabras en el número de transiciones de blanco a no blanco en lugar de en el número de transiciones de no blanco a blanco. Comprueba si tu programa funciona en toda circunstancia.
- **166** Nuestro aprendiz aventajado propone esta otra solución al problema de contar palabras:

```
1 cadena = raw_input('Escribe una frase:')
2 while cadena != '':
3     cambios = 0
4     for i in range(1, len(cadena)):
5         if cadena[i] != ' ' and cadena[i-1] == ' ':
6             cambios = cambios + 1
7
8     if cadena[-1] == ' ':
9         cambios = cambios - 1
10
11     palabras = cambios + 1
12     print 'Palabras:', palabras
13
14     cadena = raw_input('Escribe una frase:')
```

¿Es correcta?

- **167** Diseña un programa que lea una cadena y un número entero k y nos diga cuántas palabras tienen una longitud de k caracteres.
- **168** Diseña un programa que lea una cadena y un número entero k y nos diga si *alguna* de sus palabras tiene una longitud de k caracteres.
- **169** Diseña un programa que lea una cadena y un número entero k y nos diga si *todas* sus palabras tienen una longitud de k caracteres.
- **170** Escribe un programa que lea una cadena y un número entero k y muestre el mensaje «**Hay palabras largas**» si alguna de las palabras de la cadena es de longitud mayor o igual que k , y «**No hay palabras largas**» en caso contrario.
- **171** Escribe un programa que lea una cadena y un número entero k y muestre el mensaje «**Todas son cortas**» si todas las palabras de la cadena son de longitud estrictamente menor que k , y «**Hay alguna palabra larga**» en caso contrario.
- **172** Escribe un programa que lea una cadena y un número entero k y muestre el mensaje «**Todas las palabras son largas**» si todas las palabras de la cadena son de longitud mayor o igual que k , y «**Hay alguna palabra corta**» en caso contrario.
- **173** Diseña un programa que muestre la cantidad de dígitos que aparecen en una cadena introducida por teclado. La cadena 'un_1_y_un_20', por ejemplo, tiene 3 dígitos: un 1, un 2 y un 0.
- **174** Diseña un programa que muestre la cantidad de números que aparecen en una cadena leída de teclado. ¡Ojo! Con número no queremos decir dígito, sino número propiamente dicho, es decir, secuencia de dígitos. La cadena 'un_1,_un_201_y_2' por ejemplo, tiene 3 números: el 1, el 201 y el 2.

► **175** Diseña un programa que indique si una cadena leída de teclado está bien formada como número entero. El programa escribirá «Es entero» en caso afirmativo y «No es entero» en caso contrario.

Por ejemplo, para '12' mostrará «Es entero», pero para '1_2' o 'a' mostrará «No es entero».

► **176** Diseña un programa que indique si una cadena introducida por el usuario está bien formada como identificador de variable. Si lo está, mostrará el texto «Identificador válido» y si no, «Identificador inválido».

► **177** Diseña un programa que indique si una cadena leída por teclado está bien formada como número flotante.

Prueba el programa con estas cadenas: '3.1', '3.', '.1', '1e+5', '-10.2E3', '3.1e-2', '.1e01'. En todos los casos deberá indicar que se trata de números flotantes correctamente formados.

► **178** Un texto está bien parentizado si por cada paréntesis abierto hay otro más adelante que lo cierra. Por ejemplo, la cadena

'Esto_(es_(un_(ejemplo_(de)_((cadena)_bien))_parentizada)).'

está bien parentizada, pero no lo están estas otras:

'una_cadena)' '(una_cadena' '(una_(cadena)' '))una_(cadena'

Diseña un programa que lea una cadena y nos diga si la cadena está bien o mal parentizada.

► **179** Implementa un programa que lea de teclado una cadena que representa un número binario. Si algún carácter de la cadena es distinto de '0' o '1', el programa advertirá al usuario de que la cadena introducida no representa un número binario y pedirá de nuevo la lectura de la cadena.

► **180** Haz una traza para las cadenas '1101' y '010'.

► **181** Una vez más, nuestro aprendiz ha diseñado un programa diferente:

```
decimal.4.py decimal.py
1 bits = raw_input('Dame_un_número_binario:')
2
3 valor = 0
4 for bit in bits:
5     if bit == '1':
6         valor = 2 * valor + 1
7     else:
8         valor = 2 * valor
9
10 print 'Su_valor_decimal_es', valor
```

¿Es correcto? Haz trazas para las cadenas '1101' y '010'.

► **182** ¿Y esta otra versión? ¿Es correcta?

```
decimal.5.py decimal.py
1 bits = raw_input('Dame_un_número_binario:')
2
3 valor = 0
4 for bit in bits:
5     if bit == '1':
6         valor += valor + 1
7     else:
8         valor += valor
9
10 print 'Su_valor_decimal_es', valor
```

Haz trazas para las cadenas '1101' y '010'.

► **183** ¿Y esta otra? ¿Es correcta?

```
decimal.6.py decimal.py
1 bits = raw_input('Dame_un_número_binario:')
2
3 valor = 0
4 for bit in bits:
5     valor += valor + int(bit)
6
7 print 'Su_valor_decimal_es', valor
```

Haz trazas para las cadenas '1101' y '010'.

- **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 'sé₁verla₁al₁revés', 'anita₁lava₁la₁ti₁na', 'luz₁azul' y 'la₁ruta₁natural' 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?

- **201** Diseña un programa que, dados una cadena c , un índice i y un número n , muestre la subcadena de c formada por los n caracteres que empiezan en la posición de índice i .
- **202** Si a vale 'Ejemplo', ¿qué es el corte $a[:]$?
- **203** ¿Qué corte utilizarías para obtener los n caracteres de una cadena a a partir de la posición de índice i ?
- **204** Diseña un programa que, dada una cadena, muestre por pantalla todos sus prefijos. Por ejemplo, dada la cadena 'UJI', por pantalla debe aparecer:

```
U
UJ
UJI
```

- **205** Diseña un programa que lea una cadena y muestre por pantalla todas sus subcadenas de longitud 3.
- **206** Diseña un programa que lea una cadena y un entero k y muestre por pantalla todas sus subcadenas de longitud k .
- **207** Diseña un programa que lea dos cadenas a y b y nos diga si b es un prefijo de a o no.
(Ejemplo: 'sub' es un prefijo de 'subcadena'.)
- **208** Diseña un programa que lea dos cadenas a y b y nos diga si b es una subcadena de a o no.
(Ejemplo: 'de' es una subcadena de 'subcadena'.)
- **209** Diseña un programa que lea dos cadenas y devuelva el prefijo común más largo de ambas.
(Ejemplo: las cadenas 'politécnico' y 'polinización' tienen como prefijo común más largo a la cadena 'poli'.)
- **210** Diseña un programa que lea tres cadenas y muestre el prefijo común más largo de todas ellas.
(Ejemplo: las cadenas 'politécnico', 'polinización' y 'poros' tienen como prefijo común más largo a la cadena 'po'.)
- **211** El programa no funcionará bien con cualquier carta. Por ejemplo, si la variable *texto* vale 'HolaA.A.' el programa falla. ¿Por qué? ¿Sabrías corregir el programa?
- **212** Dibuja un diagrama con el estado de la memoria tras ejecutar estas sentencias:

```
>>> a = 'cadena' ↵
>>> b = a[2:3] ↵
>>> c = b + ' ' ↵
```

- **213** Dibuja diagramas que muestren el estado de la memoria paso a paso para esta secuencia de asignaciones.

```
>>> a = 'ab' ↵
>>> a *= 3 ↵
>>> b = a ↵
>>> c = a[:] ↵
>>> c = c + b ↵
```

¿Qué se mostrará por pantalla si imprimimos a , b y c al final?

- **214** ¿Qué aparecerá por pantalla al evaluar la expresión $[1][0]$? ¿Y al evaluar la expresión $[] [0]$?
- **215** Hemos asignado a x la lista $[1, 2, 3]$ y ahora queremos asignar a y una copia. Podríamos hacer $y = x[:]$, pero parece que $y = x + []$ también funciona. ¿Es así? ¿Por qué?
- **216** ¿Qué aparecerá por pantalla al ejecutar este programa?

```
1 print 'Principio'
2 for i in []:
3     print 'paso', i
4 print 'y fin'
```

- **217** ¿Qué aparecerá por pantalla al ejecutar este programa?

```
1 for i in [1] * 10:
2     print i
```

- **218** ¿Sabrías decir que resultados se mostrarán al ejecutar estas sentencias?

```
>>> [1, 2] < [1, 2] ↵
>>> [1, 2, 3] < [1, 2] ↵
>>> [1, 1] < [1, 2] ↵
>>> [1, 3] < [1, 2] ↵
>>> [10, 20, 30] > [1, 2, 3] ↵
>>> [10, 20, 3] > [1, 2, 3] ↵
>>> [10, 2, 3] > [1, 2, 3] ↵
>>> [1, 20, 30] > [1, 2, 3] ↵
>>> [0, 2, 3] <= [1, 2, 3] ↵
>>> [1] < [2, 3] ↵
>>> [1] < [1, 2] ↵
>>> [1, 2] < [0] ↵
```

► **219** Diseña un programa que tras asignar dos listas a sendas variables nos diga si la primera es menor que la segunda. No puedes utilizar operadores de comparación entre listas para implementar el programa.

► **220** ¿Qué ocurrirá al ejecutar estas órdenes Python?

```
>>> a = [1, 2, 3] ↵
>>> a is a ↵
>>> a + [] is a ↵
>>> a + [] == a ↵
```

► **221** Explica, con la ayuda de un gráfico que represente la memoria, los resultados de evaluar estas expresiones:

```
>>> a = [1, 2, 1] ↵
>>> b = [1, 2, 1] ↵
>>> (a[0] is b[0]) and (a[1] is b[1]) and (a[2] is b[2]) ↵
True
>>> a == b ↵
True
>>> a is b ↵
False
```

► **222** ¿Qué ocurrirá al ejecutar estas órdenes Python?

```
>>> [1, 2] == [1, 2] ↵
>>> [1, 2] is [1, 2] ↵
>>> a = [1, 2, 3] ↵
>>> b = [a[0], a[1], a[2]] ↵
>>> a == b ↵
>>> a is b ↵
>>> a[0] == b[1] ↵
>>> b is [b[0], b[1], b[2]] ↵
```

► **223** Que se muestra por pantalla como respuesta a cada una de estas sentencias Python:

```
>>> a = [1, 2, 3, 4, 5] ↵
>>> b = a[1:3] ↵
>>> c = a ↵
>>> d = a[:] ↵
>>> a == c ↵
>>> a == d ↵
>>> c == d ↵
>>> a == b ↵
>>> a is c ↵
>>> a is d ↵
>>> c is d ↵
>>> a is b ↵
```

► **224** Haz un programa que almacene en una variable *a* la lista obtenida con *range(1,4)* y, a continuación, la modifique para que cada componente sea igual al cuadrado del componente original. El programa mostrará la lista resultante por pantalla.

- **225** Haz un programa que almacene en *a* una lista obtenida con *range(1, n)*, donde *n* es un entero que se pide al usuario y modifique dicha lista para que cada componente sea igual al cuadrado del componente original. El programa mostrará la lista resultante por pantalla.
- **226** Haz un programa que, dada una lista *a* cualquiera, sustituya cualquier elemento negativo por cero.
- **227** ¿Qué mostrará por pantalla el siguiente programa?

```

copias.2.py
copias.py
1 a = range(0, 5)
2 b = range(0, 5)
3 c = a
4 d = b[:]
5 e = a + b
6 f = b[:1]
7 g = b[0]
8 c[0] = 100
9 d[0] = 200
10 e[0] = 300
11 print a, b, c, d, e, f, g

```

Comprueba con el ordenador la validez de tu respuesta.

- **228** Representa el estado de la memoria tras efectuar cada una de las siguientes asignaciones:

```

>>> a = [1, 2, 1] ↵
>>> b = 1 ↵
>>> c = [2, 1, 2] ↵
>>> d = c ↵
>>> d[2] = 3 ↵
>>> e = d[:1] ↵
>>> f = d[:] ↵
>>> f[0] = a[1] ↵
>>> f[1] = 1 ↵

```

- **229** Diseña un programa que construya una lista con los *n* primeros números primos (ojo: no los primos entre 1 y *n*, sino los *n* primeros números primos). ¿Necesitas usar *append*? ¿Puedes reservar en primer lugar un vector con *n* celdas nulas y asignarle a cada una de ellas uno de los números primos?
- **230** Diseña un programa que lea una lista de 10 enteros, pero asegurándose de que todos los números introducidos por el usuario son positivos. Cuando un número sea negativo, lo indicaremos con un mensaje y permitiremos al usuario repetir el intento cuantas veces sea preciso.
- **231** Diseña un programa que lea una cadena y muestre por pantalla una lista con todas sus palabras en minúsculas. La lista devuelta no debe contener palabras repetidas.
Por ejemplo: ante la cadena

'Una frase formada con palabras. Otra frase con otras palabras.'

el programa mostrará la lista

['una', 'frase', 'formada', 'con', 'palabras', 'otra', 'otras'].

Observa que en la lista no aparece dos veces la palabra «frase», aunque sí aparecía dos veces en la cadena leída.

- **232** ¿Qué sale por pantalla al ejecutar este programa?:

```

1 a = range(0, 5)
2 del a[1]
3 del a[1]
4 print a

```

- **233** Diseña un programa que elimine de una lista todos los elementos de *índice* par y muestre por pantalla el resultado. (Ejemplo: si trabaja con la lista [1, 2, 1, 5, 0, 3], ésta pasará a ser [2, 5, 3].)
- **234** Diseña un programa que elimine de una lista todos los elementos de *valor* par y muestre por pantalla el resultado. (Ejemplo: si trabaja con la lista [1, -2, 1, -5, 0, 3], ésta pasará a ser [1, 1, -5, 3].)

- **235** A nuestro programador novato se le ha ocurrido esta otra forma de eliminar el elemento de índice i de una lista a :

```
1 a = a[:i] + a[i+1:]
```

¿Funciona? Si no es así, ¿por qué? Y si funciona correctamente, ¿qué diferencia hay con respecto a usar **del** $a[i]$?

- **236** ¿Por qué este otro programa es erróneo?

```
pertenencia.2.py pertenencia.py
1 elemento = 5
2 lista = [1, 4, 5, 1, 3, 8]
3
4 for i in lista:
5     if elemento == i:
6         pertenece = True
7     else:
8         pertenece = False
9     break
10
11 if pertenece:
12     print 'Pertenece'
13 else:
14     print 'No pertenece'
```

- **237** ¿Qué hace este programa?

```
1 letra = raw_input('Dame una letra:')
2 if (len(letra) == 1 and 'a' <= letra <= 'z') or letra in ['á', 'é', 'í', 'ó', 'ú', 'ü', 'ñ']:
3     print letra, 'es una letra minúscula'
```

- **238** ¿Qué hace este programa?

```
1 letra = raw_input('Dame una letra:')
2 if len(letra) == 1 and ('a' <= letra <= 'z' or letra in 'áéíóúüñ'):
3     print letra, 'es una letra minúscula'
```

- **239** ¿Qué ocurrirá si sustituimos la primera línea de *burbuja.py* por esta otra?:

```
1 lista = ['Pepe', 'Juan', 'María', 'Ana', 'Luis', 'Pedro']
```

- **240** En una cadena llamada *texto* disponemos de un texto formado por varias frases. ¿Con qué orden simple puedes contar el número de frases?

- **241** En una cadena llamada *texto* disponemos de un texto formado por varias frases. Escribe un programa que determine y muestre el número de palabras de cada frase.

- **242** ¿Qué resulta de ejecutar esta orden?

```
>>> print ''.join(['uno', 'dos', 'tres'])
```

- **243** Disponemos de una cadena que contiene una frase cuyas palabras están separadas por un número arbitrario de espacios en blanco. ¿Podrías «estandarizar» la separación de palabras en una sola línea Python? Por estandarizar queremos decir que la cadena no empiece ni acabe con espacios en blanco y que cada palabra se separe de la siguiente por un único espacio en blanco.

- **244** Una matriz nula es aquella que sólo contiene ceros. Construye una matriz nula de 5 filas y 5 columnas.

- **245** Una matriz identidad es aquella cuyos elementos en la diagonal principal, es decir, accesibles con una expresión de la forma $M[i][i]$, valen uno y el resto valen cero. Construye una matriz identidad de 4 filas y 4 columnas.

- **246** ¿Qué resulta de ejecutar este programa?

```
1 M = [ [1, 0, 0], [0, 1, 0], [0, 0, 1] ]
2 print M[-1][0]
3 print M[-1][-1]
4 print '--'
5 for i in range(0, 3):
6     print M[i]
7     print '--'
8 for i in range(0, 3):
9     for j in range(0, 3):
10        print M[i][j]
```


► **247** ¿Qué resulta de ejecutar este programa?

```

1 M = [ [1, 0, 0], [0, 1, 0], [0, 0, 1] ]
2 s = 0.0
3 for i in range(0, 3):
4     for j in range(0, 3):
5         s += M[i][j]
6 print s / 9.0

```

► **248** Crea la siguiente matriz utilizando la técnica del bucle descrita anteriormente.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

► **249** Haz un programa que pida un entero positivo n y almacene en una variable M la matriz identidad de $n \times n$ (la que tiene unos en la diagonal principal y ceros en el resto de celdas).

► **250** Diseña un programa que lea dos matrices y calcule la diferencia entre la primera y la segunda.

► **251** Diseña un programa que lea una matriz y un número y devuelva una nueva matriz: la que resulta de multiplicar la matriz por el número. (El producto de un número por una matriz es la matriz que resulta de multiplicar cada elemento por dicho número.)

► **252** La traspuesta de una matriz A de dimensión $m \times n$ es una matriz A^T de dimensión $n \times m$ tal que $A_{i,j}^T = A_{j,i}$. Por ejemplo, si

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 12 & 6 \\ 1 & 0 & -3 \\ 10 & -1 & 0 \end{pmatrix}$$

entonces:

$$A^T = \begin{pmatrix} 1 & 2 & 1 & 10 \\ 2 & 12 & 0 & -1 \\ 3 & 6 & -3 & 0 \end{pmatrix}$$

Diseña un programa que lea una matriz y muestre su traspuesta.

► **253** Diseña un programa tal que lea una matriz A de dimensión $m \times n$ y muestre un vector v de talla n tal que

$$v_i = \sum_{j=1}^m A_{i,j},$$

para i entre 1 y n .

► **254** Diseña un programa que lea una matriz A de dimensión $m \times n$ y muestre un vector v de talla $\min(n, m)$ tal que

$$v_i = \sum_{j=1}^i \sum_{k=1}^i A_{j,k},$$

para i entre 1 y $\min(n, m)$.

► **255** Diseña un programa que determine si una matriz es prima o no. Una matriz A es prima si la suma de los elementos de cualquiera de sus filas es igual a la suma de los elementos de cualquiera de sus columnas.

► **256** Una matriz es diagonal superior si todos los elementos por debajo de la diagonal principal son nulos. Por ejemplo, esta matriz es diagonal superior:

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 12 & 6 \\ 0 & 0 & -3 \\ 0 & 0 & 0 \end{pmatrix}$$

Diseña un programa que diga si una matriz es o no es diagonal superior.

► **257** ¿Funciona esta otra forma de contar los vecinos de la casilla de la fila y y columna x ?

```

n = -tablero[y][x]
for i in [-1, 0, 1]:
    for j in [-1, 0, 1]:
        if y+i >= 0 and y+i < filas and x+j >= 0 and x+j < columnas:
            n += tablero[y+i, x+j]

```

► **258** El «juego de la vida parametrizado» es una generalización del juego de la vida. En él, el número de vecinos vivos necesarios para activar las reglas de nacimiento, supervivencia, aislamiento y superpoblación están parametrizados. Haz un programa que solicite al usuario el número de células vecinas vivas necesarias para que se disparen las diferentes reglas y muestre cómo evoluciona el tablero con ellas.

► **259** El juego de la vida toroidal se juega sobre un tablero de dimensión finita $m \times n$ con unas reglas de vecindad diferentes. Una casilla de coordenadas (y, x) tiene siempre 8 vecinas, aunque esté en un borde:

$((y-1) \bmod m, (x-1) \bmod n)$	$((y-1) \bmod m, x)$	$((y-1) \bmod m, (x+1) \bmod n)$
$(y, (x-1) \bmod n)$		$(y, (x+1) \bmod n)$
$((y+1) \bmod m, (x-1) \bmod n)$	$((y+1) \bmod m, x)$	$((y+1) \bmod m, (x+1) \bmod n)$

donde mod es el operador módulo (en Python, %).

Implementa el juego de la vida toroidal en el entorno PythonG.

► **260** El juego de la vida es un tipo particular de *autómata celular bidimensional*. Hay autómatas celulares unidimensionales. En ellos, una lista de valores (en su versión más simple, ceros y unos) evoluciona a lo largo del tiempo a partir del estado de sus celdas vecinas (solo las celdas izquierda y derecha en su versión más simple) y de ella misma en el instante anterior.

Por ejemplo, una regla $001 \rightarrow 1$ se lee como «la célula está viva si en la iteración anterior estaba muerta y tenía una célula muerta a la izquierda y una célula viva a la derecha». Una especificación completa tiene este aspecto:

000 \rightarrow 0 001 \rightarrow 1 010 \rightarrow 1 011 \rightarrow 0 100 \rightarrow 1 101 \rightarrow 1 110 \rightarrow 0 111 \rightarrow 0

Y aquí tienes una representación (usando asteriscos para los unos y puntos para los ceros) de la evolución del sistema durante sus primeros pulsos partiendo de una configuración muy sencilla (un solo uno):

```
Pulso 0 : . . . . . * . . . . .
Pulso 1 : . . . . . * * * . . . . .
Pulso 2 : . . . . . * . . * . . . .
Pulso 3 : . . . . . * * * . * * . .
Pulso 4 : . . . . . * . . * . . * . .
Pulso 5 : . . . . . * * * . * * . * *
Pulso 6 : . . . . * . . . * . . * . .
```

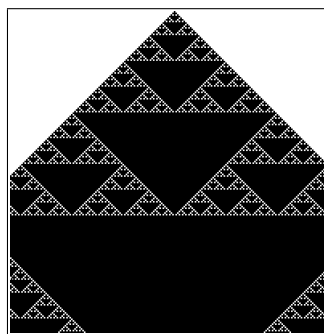
Implementa un programa para estudiar la evolución de autómatas celulares unidimensionales. El programa leerá un conjunto de reglas por teclado y un número de pulsos. A continuación, mostrará en el terminal de texto la evolución del autómata partiendo de una configuración con sólo una celda viva que ocupa la posición central del universo.

Cuando tengas el programa, explora las siguientes reglas:

- 000 \rightarrow 0 001 \rightarrow 1 010 \rightarrow 1 011 \rightarrow 1 100 \rightarrow 1 101 \rightarrow 0 110 \rightarrow 0 111 \rightarrow 0
- 000 \rightarrow 0 001 \rightarrow 0 010 \rightarrow 1 011 \rightarrow 1 100 \rightarrow 1 101 \rightarrow 0 110 \rightarrow 0 111 \rightarrow 0
- 000 \rightarrow 0 001 \rightarrow 1 010 \rightarrow 1 011 \rightarrow 1 100 \rightarrow 0 101 \rightarrow 1 110 \rightarrow 1 111 \rightarrow 0
- 000 \rightarrow 0 001 \rightarrow 1 010 \rightarrow 1 011 \rightarrow 1 100 \rightarrow 0 101 \rightarrow 1 110 \rightarrow 1 111 \rightarrow 0
- 000 \rightarrow 0 001 \rightarrow 1 010 \rightarrow 1 011 \rightarrow 0 100 \rightarrow 1 101 \rightarrow 1 110 \rightarrow 0 111 \rightarrow 1

► **261** Modifica el programa del ejercicio anterior para obtener una representación gráfica en PythonG. Tradicionalmente se muestra en cada fila el estado del «tablero unidimensional» en cada pulso. Así se puede estudiar mejor la evolución del autómata.

Aquí tienes lo que debería mostrar tu programa para el último juego de reglas del ejercicio anterior:



► **262** Define una función llamada *raiz_cubica* que devuelva el valor de $\sqrt[3]{x}$.

(Nota: recuerda que $\sqrt[3]{x}$ es $x^{1/3}$ y ándate con ojo, no sea que utilices una división entera y eleves x a la potencia 0, que es el resultado de calcular $1/3$.)

- **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.)

- **317** Diseña un programa que permita jugar a dos personas al tres en raya.
- **318** Diseña un programa que permita jugar al tres en raya enfrentando a una persona al ordenador. Cuando el ordenador empiece una partida, debe ganarla siempre. (Este ejercicio es difícil. Si no conoces la estrategia ganadora, búscala en Internet.)
- **319** Diseña un programa que permita que dos personas jueguen a las damas. El programa debe verificar que todos los movimientos son válidos.
- **320** Diseña un programa que permita que dos personas jueguen al ajedrez. El programa debe verificar que todos los movimientos son válidos.
- **321** Haz una traza de *area_y_angulo.py* al solicitar el valor del ángulo opuesto al lado de longitud 5 en un triángulo de lados con longitudes 5, 4 y 3.
- **322** ¿Qué aparecerá por pantalla al ejecutar el siguiente programa?

```

triangulo.2.py
1 from math import sqrt
2
3 def area_triangulo(a, b, c):
4     s = (a + b + c) / 2.0
5     return sqrt(s * (s-a) * (s-b) * (s-c))
6
7 s = 4
8 print area_triangulo(s-1, s, s+1)
9 print s
10 print a

```

► **323** La función *area_triangulo* que hemos definido puede provocar un error en tiempo de ejecución: si el argumento de la raíz cuadrada calculada en su última línea es un número negativo, se producirá un error de dominio. Haz que la función sólo llame a *sqrt* si su argumento es mayor o igual que cero. Si el argumento es un número negativo, la función debe devolver el valor cero. Detecta también posibles problemas en *angulo_alfa* y modifica la función para evitar posibles errores al ejecutar el programa.

► **324** Vamos a adquirir una vivienda y para eso necesitaremos una hipoteca. La cuota mensual *m* que hemos de pagar para amortizar una hipoteca de *h* euros a lo largo de *n* años a un interés compuesto del *i* por cien anual se calcula con la fórmula:

$$m = \frac{hr}{1 - (1 + r)^{-12n}},$$

donde $r = i/(100 \cdot 12)$. Define una función que calcule la cuota (redondeada a dos decimales) dados *h*, *n* e *i*. Utiliza cuantas variables locales consideres oportuno, pero al menos *r* debe aparecer en la expresión cuyo valor se devuelve y antes debe calcularse y almacenarse en una variable local.

Nota: puedes comprobar la validez de tu función sabiendo que hay que pagar la cantidad de 1 166.75 € al mes para amortizar una hipoteca de 150 000 € en 15 años a un interés del 4.75% anual.

► **325** Diseña una función que nos devuelva la cantidad de euros que habremos pagado finalmente al banco si abrimos una hipoteca de *h* euros a un interés del *i* por cien en *n* años. Si te conviene, puedes utilizar la función que definiste en el ejercicio anterior.

Nota: con los datos del ejemplo anterior, habremos pagado un total de 210 015 €.

► **326** Diseña una función que nos diga qué cantidad *de intereses* (en euros) habremos pagado finalmente al banco si abrimos una hipoteca de *h* euros a un interés del *i* por cien en *n* años. Si te conviene, puedes utilizar las funciones que definiste en los ejercicios anteriores.

Nota: con los datos del ejemplo anterior, habremos pagado un total de $210\,015 - 150\,000 = 60\,015$ € en intereses.

► **327** Diseña una función que nos diga qué tanto por cien del capital inicial deberemos pagar en intereses al amortizar completamente la hipoteca. Si te conviene, puedes utilizar las funciones que definiste en los ejercicios anteriores.

Nota: con los datos del ejemplo anterior, habremos pagado un interés total del 40.01% (60 015 € es el 40.01% de 150 000 €).

► **328** Diseña un *procedimiento* que muestre por pantalla la cuota mensual que corresponde pagar por una hipoteca para un capital de *h* euros al *i*% de interés anual durante 10, 15, 20 y 25 años. (Si te conviene, rescata ahora las funciones que diseñaste como solución de los ejercicios anteriores.)

► **329** Diseña un *procedimiento* que muestre por pantalla el capital total pagado al banco por una hipoteca de *h* euros al *i*% de interés anual durante 10, 15, 20 y 25 años. (Si te conviene, rescata ahora las funciones que diseñaste como solución de los ejercicios anteriores.)

► **330** Diseña una función que reciba dos listas y devuelva los elementos comunes a ambas, sin repetir ninguno (intersección de conjuntos).

Ejemplo: si recibe las listas [1, 2, 1] y [2, 3, 2, 4], devolverá la lista [2].

► **331** Diseña una función que reciba dos listas y devuelva los elementos que pertenecen a una o a otra, pero sin repetir ninguno (unión de conjuntos).

Ejemplo: si recibe las listas [1, 2, 1] y [2, 3, 2, 4], devolverá la lista [1, 2, 3, 4].

► **332** Diseña una función que reciba dos listas y devuelva los elementos que pertenecen a la primera pero no a la segunda, sin repetir ninguno (diferencia de conjuntos).

Ejemplo: si recibe las listas [1, 2, 1] y [2, 3, 2, 4], devolverá la lista [1].

► **333** Diseña una función que, dada una lista de números, devuelva otra lista que sólo incluya sus números impares.

► **334** Diseña una función que, dada una lista de nombres y una letra, devuelva una lista con todos los nombres que empiezan por dicha letra.

► **335** Diseña una función que, dada una lista de números, devuelva otra lista con sólo aquellos números de la primera que son primos.

► **336** Diseña una función que, dada una lista de números, devuelva una lista con todos los pares de números que podemos formar con uno de la primera lista y otro de la segunda. Por ejemplo, si se suministran las listas [1, 3, 5] y [2, 5], la lista resultante es

[1, 2], [1, 5], [3, 2], [3, 5], [5, 2], [5, 5]].

► **337** Diseña una función que, dada una lista de números, devuelva una lista con todos los pares de números *amigos* que podemos formar con uno de la primera lista y otro de la segunda.

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

```

parametros.2.py      parametros.py
1 def incrementa(a):
2     a = a + 1
3     return a
4
5 a = 1
6 b = incrementa(a)
7
8 print 'a:', a
9 print 'b:', b

```

Hazte un dibujo del estado de la pila de llamadas paso a paso para entender bien qué está pasando al ejecutar cada sentencia.

► **339** ¿Qué mostrará por pantalla el siguiente programa al ejecutarse?

```

ejercicio.parametros.4.py      ejercicio_parametros.py
1 def modifica(a, b):
2     for elemento in b:
3         a.append(elemento)
4     b = b + [4]
5     a[-1] = 100
6     del b[0]
7     return b[:]
8
9 lista1 = [1, 2, 3]
10 lista2 = [1, 2, 3]
11
12 lista3 = modifica(lista1, lista2)
13
14 print lista1
15 print lista2
16 print lista3

```

► **340** ¿Qué muestra por pantalla este programa al ser ejecutado?

```

ejercicio.parametros.5.py      ejercicio_parametros.py
1 def modifica_parametros(x, y):
2     x = 1
3     y[0] = 1
4

```




```

5 a = 0
6 b = [0, 1, 2]
7 modifica_parametros(a, b)
8
9 print a
10 print b

```

► **341** ¿Qué muestra por pantalla este programa al ser ejecutado?

 ejercicio_parametros.6.py ejercicio_parametros.py

```

1 def modifica_parametros(x, y):
2     x = 1
3     y.append(3)
4     y = y + [4]
5     y[0] = 10
6
7 a = 0
8 b = [0, 1, 2]
9 modifica_parametros(a, b)
10 print a
11 print b

```

► **342** Utiliza las funciones desarrolladas en el ejercicio **307** y diseña nuevas funciones para construir un programa que presente el siguiente menú y permita ejecutar las acciones correspondientes a cada opción:


```

1) Añadir estudiante y calificación
2) Mostrar lista de estudiantes con sus calificaciones
3) Calcular la media de las calificaciones
4) Calcular el número de aprobados
5) Mostrar los estudiantes con mejor calificación
6) Mostrar los estudiantes con calificación superior a la media
7) Consultar la nota de un estudiante determinado
8) FINALIZAR EJECUCIÓN DEL PROGRAMA

```

► **343** ¿Qué ocurre con el elemento central de la lista cuando la lista tiene un número impar de elementos? ¿Nuestra función invierte correctamente la lista?

► **344** Un aprendiz sugiere esta otra solución. ¿Funciona?


 inversion.2.py inversion.py

```

1 def invierte(lista):
2     for i in range(len(lista)/2):
3         c = lista[i]
4         lista[i] = lista[-i-1]
5         lista[-i-1] = c

```

► **345** ¿Qué muestra por pantalla este programa al ser ejecutado?


 abslista.2.py abslista.py

```

1 def abs_lista(lista):
2     for i in range(len(lista)):
3         lista[i] = abs(lista[i])
4
5 milista = [1, -1, 2, -3, -2, 0]
6 abs_lista(milista)
7 print milista

```

► **346** ¿Qué mostrará por pantalla el siguiente programa al ejecutarse?

 intercambio.2.py intercambio.py

```

1 def intento_de_intercambio(a, b):
2     aux = a
3     a = b
4     b = aux
5
6 lista1 = [1, 2]
7 lista2 = [3, 4]
8

```

```

9 intento_de_intercambio(lista1, lista2)
10
11 print lista1
12 print lista2

```

- **347** Diseña un procedimiento que, dada una lista de números, la modifique para que sólo sobrevivan a la llamada aquellos números que son perfectos.
- **348** Diseña una función *duplica* que reciba una lista de números y la modifique duplicando el valor de cada uno de sus elementos. (Ejemplo: la lista [1, 2, 3] se convertirá en [2, 4, 6].)
- **349** Diseña una función *duplica_copia* que reciba una lista de números y devuelva *otra* lista en la que cada elemento sea el doble del que tiene el mismo índice en la lista original. La lista original *no debe sufrir ninguna modificación* tras la llamada a *duplica_copia*.
- **350** Diseña una función que reciba una lista y devuelva otra lista cuyo contenido sea el resultado de concatenar la lista original consigo misma. La lista original no debe modificarse.
- **351** Diseña una función que reciba una lista y devuelva otra lista cuyo contenido sea la lista original, pero con sus componentes en orden inverso. La lista original no debe modificarse.
- **352** Diseña una función que reciba una lista y devuelva una copia de la lista concatenada con una inversión de sí misma. Puedes utilizar, si lo consideras conveniente, funciones que has desarrollado en ejercicios anteriores.
- **353** Diseña una función que reciba una lista y devuelva una lista cuyo contenido sea la lista original concatenada con una versión invertida de ella misma. La lista original no debe modificarse.
- **354** Diseña una función que reciba una lista y devuelva una copia de la lista con sus elementos ordenados de menor a mayor. La lista original no debe modificarse.
- **355** Diseña un procedimiento que reciba una lista y ordene sus elementos de menor a mayor.
- **356** Diseña una función que reciba una matriz y, si es cuadrada (es decir, tiene igual número de filas que de columnas), devuelva la suma de todos los componentes dispuestos en la diagonal principal (es decir, todos los elementos de la forma $A_{i,i}$). Si la matriz no es cuadrada, la función devolverá *None*.
- **357** Guardamos en una matriz de $m \times n$ elementos la calificación obtenida por m estudiantes (a los que conocemos por su número de lista) en la evaluación de n ejercicios entregados semanalmente (cuando un ejercicio no se ha entregado, la calificación es -1).
- Diseña funciones y procedimientos que efectúen los siguiente cálculos:
- Dado el número de un alumno, devolver el número de ejercicios entregados.
 - Dado el número de un alumno, devolver la media sobre los ejercicios entregados.
 - Dado el número de un alumno, devolver la media sobre los ejercicios entregados si los entregó todos; en caso contrario, la media es 0.
 - Devolver el número de todos los alumnos que han entregado todos los ejercicios y tienen una media superior a 3.5 puntos.
 - Dado el número de un ejercicio, devolver la nota media obtenida por los estudiantes que lo presentaron.
 - Dado el número de un ejercicio, devolver la nota más alta obtenida.
 - Dado el número de un ejercicio, devolver la nota más baja obtenida.
 - Dado el número de un ejercicio, devolver el número de estudiantes que lo han presentado.
 - Devolver el número de abandonos en función de la semana. Consideramos que un alumno abandonó en la semana x si no ha entregado ningún ejercicio desde entonces. Este procedimiento mostrará en pantalla el número de abandonos para cada semana (si un alumno no ha entregado nunca ningún ejercicio, abandonó en la «semana cero»).
- **358** Hay dos ocasiones en las que se devuelve la lista [0, 0, 0]. ¿Puedes modificar el programa para que sólo se devuelva esa lista explícita desde un punto del programa?
- **359** ¿Y si a pudiera tomar valores enteros negativos? Diseña una función *exponencial* que trate también ese caso. (Recuerda que $e^{-a} = 1/e^a$.)

- **360** ¿Es correcta esta otra versión? (Hemos destacado los cambios con respecto a la última.)

```

exponencial.2.py
def elevado(a, k):
    productorio = 1.0
    for i in range(k):
        productorio *= a
    return productorio

def factorial(k):
    productorio = 1.0
    for i in range(2, k):
        productorio *= i
    return productorio

def exponencial(a, n):
    sumatorio = 0.0
    for k in range(n):
        sumatorio += elevado(a, k) / factorial(k)
    return sumatorio

```

- **361** Las funciones seno y coseno se pueden calcular así

$$\begin{aligned}\sin(x) &= x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!} \\ \cos(x) &= 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!}\end{aligned}$$

Diseña sendas funciones *seno* y *coseno* para aproximar, respectivamente, el seno y el coseno de x con n términos del sumatorio correspondiente.

- **362** Diseña un programa similar que muestre el valor de *factorial*(n) para n entre 0 y 7.

- **363** Modifica las funciones que has propuesto como solución al ejercicio **361** aprovechando las siguientes relaciones, válidas para n mayor que 0:

$$\begin{aligned}\frac{(-1)^n x^{2n+1}}{(2n+1)!} &= -\frac{x^2}{(n+1) \cdot n} \cdot \frac{(-1)^{n-1} x^{2n-1}}{(2n-1)!}, \\ \frac{(-1)^n x^{2n}}{(2n)!} &= -\frac{x^2}{n \cdot (n-1)} \cdot \frac{(-1)^{n-1} x^{2n}}{(2n)!}.\end{aligned}$$

Cuando n vale 0, tenemos:

$$\frac{(-1)^0 x^1}{1!} = x, \quad \frac{(-1)^0 x^0}{0!} = 1.$$

- **364** Modifica la función *exponencial2* del programa anterior para que no se efectúen las ineficientes llamadas a *elevado* y *factorial*.

- **365** La función *biseccion* aún no está acabada del todo. ¿Qué ocurre si el usuario introduce un intervalo $[a, b]$ tal que $f(a)$ y $f(b)$ tienen el mismo signo? ¿Y si $f(a)$ o $f(b)$ valen 0? Modifica la función para que sólo inicie la búsqueda cuando procede y, en caso contrario, devuelva el valor especial *None*. Si $f(a)$ o $f(b)$ valen cero, *biseccion* devolverá el valor de a o b , según proceda.

- **366** Modifica el programa para que solicite al usuario los valores a , b y ϵ . El programa sólo aceptará valores de a y b tales que $a < b$.

- **367** Haz una traza de la pila de llamadas a función paso a paso para *factorial*(5).

- **368** También podemos formular recursivamente la suma de los n primeros números naturales:

$$\sum_{i=1}^n i = \begin{cases} 1, & \text{si } n = 1; \\ n + \sum_{i=1}^{n-1} i, & \text{si } n > 1. \end{cases}$$

Diseña una función Python recursiva que calcule el sumatorio de los n primeros números naturales.

- **369** Inspirándote en el ejercicio anterior, diseña una función recursiva que, dados m y n , calcule

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

- **370** La siguiente función implementa recursivamente una comparación entre dos números naturales. ¿Qué comparación?

```
compara.py
def comparacion(a, b):
    1  if b == 0:
    2      return False
    3  elif a == 0:
    4      return True
    5  else:
    6      return comparacion(a-1, b-1)
    7
```

- **371** Dibuja un árbol de llamadas que muestre paso a paso lo que ocurre cuando calculas *bits*(63).
- **372** Diseña una función recursiva que calcule el número de dígitos que tiene un número entero (en base 10).
- **373** Calcula F_{12} con ayuda de la función que hemos definido.
- **374** Dibuja el árbol de llamadas para *fibonacci*(5).
- **375** Modifica la función para que, cada vez que se la llame, muestre por pantalla un mensaje que diga «Empieza cálculo de Fibonacci de n », donde n es el valor del argumento, y para que, justo antes de acabar, muestre por pantalla «Acaba cálculo de Fibonacci de n y devuelve el valor m », donde m es el valor a devolver. A continuación, llama a la función para calcular el cuarto número de Fibonacci y analiza el texto que aparece por pantalla. Haz lo mismo para el décimo número de Fibonacci.
- **376** Puedes calcular recursivamente los números combinatorios sabiendo que, para $n \geq m$,

$$\binom{n}{m} = \binom{n-1}{m} + \binom{n-1}{m-1}$$

y que

$$\binom{n}{n} = \binom{n}{0} = 1.$$

Diseña un programa que, a partir de un valor n leído de teclado, muestre $\binom{n}{m}$ para m entre 0 y n . El programa llamará a una función *combinaciones* definida recursivamente.

- **377** El número de formas diferentes de dividir un conjunto de n números en k subconjuntos se denota con $\left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\}$ y se puede definir recursivamente así:

$$\left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\} = \left\{ \begin{smallmatrix} n-1 \\ k-1 \end{smallmatrix} \right\} + k \left\{ \begin{smallmatrix} n-1 \\ k \end{smallmatrix} \right\}$$

El valor de $\left\{ \begin{smallmatrix} n \\ 1 \end{smallmatrix} \right\}$, al igual que el de $\left\{ \begin{smallmatrix} n \\ n \end{smallmatrix} \right\}$, es 1. Diseña un programa que, a partir de un valor n leído de teclado, muestre $\left\{ \begin{smallmatrix} n \\ m \end{smallmatrix} \right\}$ para m entre 0 y n . El programa llamará a una función *particiones* definida recursivamente.

- **378** Haz una traza de las llamadas a *mcd* para los números 1470 y 693.
- **379** Haz una traza de las llamadas a *mcd* para los números 323 y 323.
- **380** En el apartado ?? presentamos el método de la bisección. Observa que, en el fondo, se trata de un método recursivo. Diseña una función que implemente el método de la bisección recursivamente.
- **381** Es hora de echar una manita a los monjes. Ellos han de resolver el problema con 64 discos. ¿Por qué no pruebas a ejecutar *resuelve_hanoi*(64, 1, 3, 2)?
- **382** ¿Cuántos movimientos son necesarios para resolver el problema de las torres de Hanoi con 1 disco, y con 2, y con 3, ...? Diseña una función *movimientos_hanoi* que reciba un número y devuelva el número de movimientos necesarios para resolver el problema de la torres de Hanoi con ese número de discos.
- **383** Implementa un programa en el entorno PythonG que muestre gráficamente la resolución del problema de las torres de Hanoi.
- **384** Dibuja el árbol de llamadas para *resuelve_hanoi*(4, 1, 3, 2).

► **385** Puedes jugar con los diferentes parámetros que determinan la curva de von Kock para obtener infinidad de figuras diferentes. Te mostramos algunas de ellas junto a las nuevas expresiones de cálculo de los puntos (x_c, y_c) , (x_d, y_d) y (x_e, y_e) :

```

7 xc = xa + (xb - xa) / 3.0
8 yc = ya + (yb - ya) / 3.0
9 xd = xb + (xa - xb) / 3.0
10 yd = yb + (ya - yb) / 3.0
11 xe = (xc+xd)*cos(pi/4) - (yd-yc)*sin(pi/3)
12 ye = (yc+yd)*cos(pi/4) + (xd-xc)*sin(pi/3)

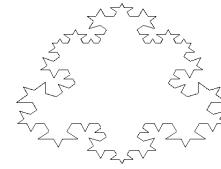
```



```

7 xc = xa + (xb - xa) / 3.0
8 yc = ya + (yb - ya) / 3.0
9 xd = xb + (xa - xb) / 3.0
10 yd = yb + (ya - yb) / 3.0
11 xe = (xc+xd)*cos(pi/3) - 2*(yd-yc)*sin(pi/3)
12 ye = (yc+yd)*cos(pi/3) + (xd-xc)*sin(pi/3)

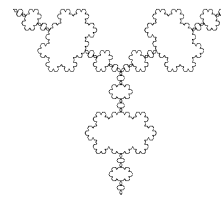
```



```

7 xc = xa + (xb - xa) / 3.0
8 yc = ya + (yb - ya) / 3.0
9 xd = xb + (xa - xb) / 3.0
10 yd = yb + (ya - yb) / 3.0
11 xe = (xc+xd)*cos(pi/3) + (yd-yc)*sin(pi/3)
12 ye = (yc+yd)*cos(pi/3) - (xd-xc)*sin(pi/3)

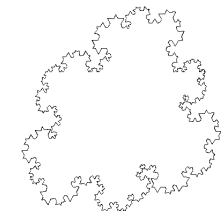
```



```

7 xc = xa + (xb - xa) / 3.0
8 yc = ya + (yb - ya) / 4.0
9 xd = xb + (xa - xb) / 5.0
10 yd = yb + (ya - yb) / 3.0
11 xe = (xc+xd)*cos(pi/3) - (yd-yc)*sin(pi/3)
12 ye = (yc+yd)*cos(pi/3) + (xd-xc)*sin(pi/3)

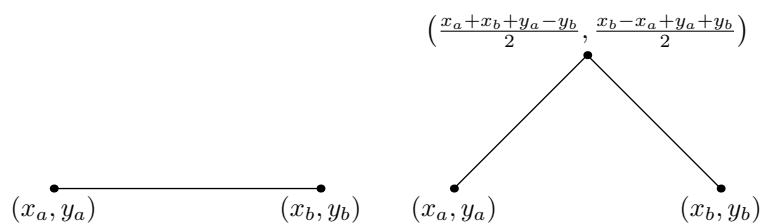
```



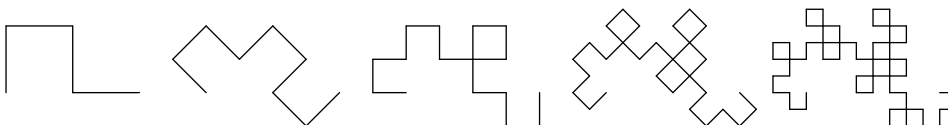
Prueba a cambiar los diferentes parámetros y trata de predecir la figura que obtendrás en cada caso antes de ejecutar el programa.

(Recuerda definir adecuadamente las coordenadas con *window_coordinates* para que te quepan las figuras.)

► **386** La curva dragón se define de modo aún más sencillo que la curva de von Koch. La curva dragón de nivel 0 que une los puntos (x_a, y_a) y (x_b, y_b) es la línea recta que las une. La curva dragón de nivel 1 entre (x_a, y_a) y (x_b, y_b) se forma con dos curvas dragón de nivel 0: la que une (x_a, y_a) con $(\frac{x_a+x_b+y_a-y_b}{2}, \frac{x_b-x_a+y_a+y_b}{2})$ y la que une (x_b, y_b) con $(\frac{x_a+x_b+y_a-y_b}{2}, \frac{x_b-x_a+y_a+y_b}{2})$. He aquí las curvas dragón de niveles 0 y 1:



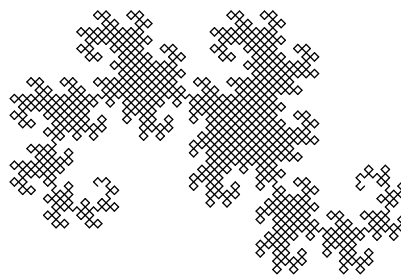
Ya ves cuál es el principio recursivo con el que se generan curvas dragón. Aquí tienes las curvas dragón de niveles 2, 3, 4, 5 y 6.



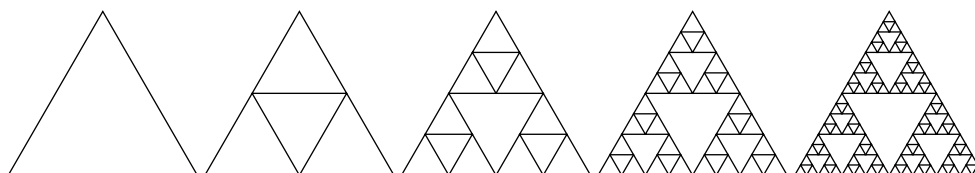
El perfil de las curvas dragón tiene una analogía con las dobleces de una hoja de papel. La curva dragón de nivel 0 es el perfil de una hoja de papel que no ha sido doblada. La de nivel 1 ha sido doblada una vez y desdoblada hasta que las partes dobladas forman ángulos de 90 grados. La curva dragón de nivel 1 es el perfil de una hoja doblada dos veces y desdoblada de forma que cada parte forme un ángulo de 90 grados con la siguiente.

Diseña un programa que dibuje, en el entorno PythonG, curvas dragón entre dos puntos del nivel que se desee.

Por cierto, ¿de dónde viene el nombre de «curva dragón»? Del aspecto que presenta en niveles «grandes». Aquí tienes la curva dragón de nivel 11:



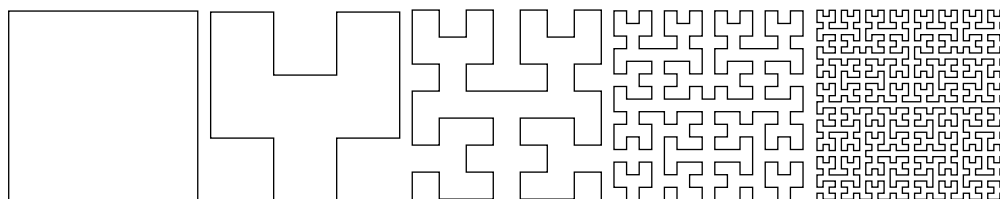
► **387** Otra figura recursiva que es todo un clásico es la criba o triángulo de Sierpinski. En cada nivel de recursión se divide cada uno de los triángulos del nivel anterior en tres nuevos triángulos. Esta figura muestra los triángulos de Sierpinski para niveles de recursión de 0 a 4:



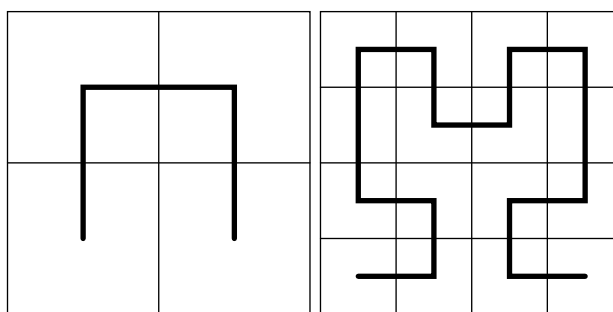
Diseña un programa para PythonG que dibuje triángulos de Sierpinski para un nivel de recursión dado.

(Por cierto, ¿no te parecen los triángulos de Sierpinski sospechosamente similares a la figura del ejercicio 261?)

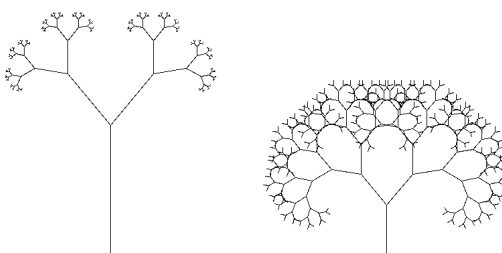
► **388** Otra curva fractal de interés es la denominada «curva de relleno del espacio de Hilbert». Esta figura te muestra dicha curva con niveles de recursión 0, 1, 2, 3 y 4:

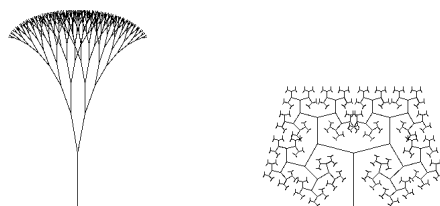


Diseña un programa capaz de dibujar curvas de relleno del espacio de Hilbert en el entorno PythonG dado el nivel de recursión deseado. Estas figuras te pueden ser de ayuda para descubrir el procedimiento de cálculo que has de programar:



► **389** Un curiosa aplicación de la recursión es la generación de paisajes por ordenador. Las montañas, por ejemplo, se dibujan con modelos recursivos: los denominados fractales (las curvas de von Koch, entre otras, son fractales). Los árboles pueden generarse también con procedimientos recursivos. Estas imágenes, por ejemplo, muestran «esqueletos» de árboles generados en el entorno PythonG:





Todos han sido generados con una misma función recursiva, pero usando diferentes argumentos. Te vamos a describir el principio básico de generación de estos árboles, pero has de ser tú mismo quien diseñe una función recursiva capaz de efectuar este tipo de dibujos. Vamos con el método. El usuario nos proporciona los siguientes datos:

- Los puntos en los que empieza y acaba el tronco.
- El ángulo α que forma la rama que parte a mano derecha del tronco con el propio tronco. La rama que parte a mano izquierda lo hace con un ángulo $-\alpha$.
- La proporción (en tanto por uno) del tamaño de las ramas con respecto al tronco.
- El nivel de recursión deseado.

La recursión tiene lugar cuando consideramos que cada una de las dos ramas es un nuevo tronco.

Por cierto, los árboles ganan bastante si en primeros niveles de recursión usas un color anaranjado o marrón y en los últimos usas un color verde.

► **390** Los árboles que hemos generado en el ejercicio anterior parecen un tanto artificiales por ser tan regulares y simétricos. Introducir el azar en su diseño los hará parecer más naturales. Modifica la función del ejercicio anterior para que tanto el ángulo como la proporción rama/tronco se escojan aleatoriamente (dentro de ciertos márgenes).

Aquí tienes un par de ejemplos. El árbol de la izquierda sí parece bastante real y el de la derecha parece mecido por el viento (bueno, ¡más bien por un huracán!).



► **391** Construye un módulo llamado *dni* que incluya las funciones propuestas en los ejercicios 270 y 296.

Usa el módulo desde un programa que pida al usuario su número de DNI y su letra. Si el usuario mete un número y letra de DNI correctos, el programa emitirá el mensaje «Bienvenido». En caso contrario dirá «Ha cometido ud. un error».

► **392** Diseña un módulo que agrupe las funciones relacionadas con hipotecas de los ejercicios 324–327. Documenta adecuadamente el módulo.

► **393** Diseña un módulo similar al anterior pero que permita efectuar cálculos con vectores n -dimensionales, donde n es un valor arbitrario. Las funciones que debes definir son:

- *v_lee_vector*: Pide el valor de n y a continuación lee los n componentes del vector. El resultado devuelto es la lista de los componentes.
- *v_muestra_vector*: Muestra por pantalla el vector en la notación (v_1, v_2, \dots, v_n) .
- *v_longitud*: devuelve la longitud del vector, que es

$$\sqrt{\sum_{i=1}^n v_i^2}$$

- *v_suma*: Devuelve la suma de dos vectores. Los dos vectores deben tener la misma dimensión. Si no la tienen, *v_suma* devolverá el valor *None*.

- *v_producto_escalar*: Devuelve el producto escalar de dos vectores. Los dos vectores deben tener la misma dimensión. Si no la tienen, la función devolverá el valor *None*.

► **394** Diseña un módulo que facilite el trabajo con conjuntos. Recuerda que un conjunto es una lista en la que no hay elementos repetidos. Deberás implementar las siguientes funciones:

- *lista_a_conjunto(lista)*: Devuelve un conjunto con los mismos elementos que hay en *lista*, pero sin repeticiones. (Ejemplo: *lista_a_conjunto*([1,1,3,2,3]) devolverá la lista [1, 2, 3] (aunque también se acepta como equivalente cualquier permutación de esos mismos elementos, como [3,1,2] o [3,2,1]).
- *union(A, B)*: devuelve el conjunto resultante de unir los conjuntos *A* y *B*.
- *interseccion(A, B)*: devuelve el conjunto cuyos elementos pertenecen a *A* y a *B*.
- *diferencia(A, B)*: devuelve el conjunto de elementos que pertenecen a *A* y no a *B*.
- *iguales(A, B)*: devuelve cierto si ambos conjuntos tienen los mismos elementos, y falso en caso contrario. (Nota: ten en cuenta que los conjuntos representados por las listas [1, 3, 2] y [2, 1, 3] son iguales.)

► **395** ¿Es correcta esta otra versión de la función *suma*?

```
1 def suma(a, b):
2     c = []
3     m = minimo(len(a), len(b))
4     for i in range(m):
5         c.append(a[i] + b[i])
6     c = c + a[m:] + b[m:]
7     return c
```

► **396** Diseña el siguiente programa que usa el módulo *polinomios* y, si te parece conveniente, enriquece dicho módulo con nuevas funciones útiles para el manejo de polinomios. El programa presentará al usuario este menú:

```
1) Leer polinomio a
2) Mostrar polinomio a
3) Leer polinomio b
4) Mostrar polinomio b
5) Sumar polinomios a y b
6) Restar a de b
7) Restar b de a
8) Multiplicar a por b
9) FIN DE PROGRAMA
```

► **397** ¿Funcionan bien las funciones que hemos definido cuando suministramos listas vacías? Corrige las funciones para que traten correctamente este caso particular.

► **398** Enriquece el módulo *estadisticas* añadiendo una función que calcule el coeficiente de variación (definido como σ/\bar{a}) y el recorrido de la lista (que es la diferencia entre el mayor y el menor elemento de la lista).

► **399** Suponiendo que nos suministran una lista de enteros, diseña una función que calcule su moda. La moda es el elemento más repetido en una serie de valores.

► **400** Diseña una función llamada *es_cuadrada* que devuelva *True* si la matriz es cuadrada (tiene igual número de filas que columnas) y *False* en caso contrario. Sírrete de la función *dimension* para averiguar la dimensión de la matriz.

► **401** Enriquece el módulo *matrices.py* con una función que devuelva el producto de dos matrices. Si las matrices no son «multiplicables», la función devolverá *None*.

► **402** Modifica el programa del ejercicio anterior enriqueciendo el tipo de datos *Persona* con un nuevo campo: el sexo, que codificaremos con una letra ('M' para mujer y 'V' para varón). Modifica la función *mostrar_persona* para que también imprima el valor del nuevo campo.

► **403** Diseña una función que permita determinar si una persona es menor de edad y devuelva cierto si la edad es menor que 18, y falso en caso contrario.

► **404** Diseña una función *nombre_de_pila* que devuelva el nombre de pila de una *Persona*. Supondremos que el nombre de pila es la primera palabra del campo *nombre* (es decir, que no hay nombres compuestos).

► **405** Diseña un programa que pida por teclado los datos de varias personas y los añada a una lista inicialmente vacía. Cada vez que se lean los datos de una persona el programa preguntará si se desea continuar introduciendo nuevas personas. Cuando el usuario responda que no, el programa se detendrá.

► **406** Modifica el programa del ejercicio anterior para que, a continuación, muestre el nombre de la persona más vieja. Si dos o más personas coinciden en tener la mayor edad, el programa mostrará el nombre de todas ellas.

► **407** ¿Qué mostrará por pantalla la ejecución del siguiente programa?

```

ejercicio_registros.py
ejercicio_registros.py
1 from record import record
2
3 class Persona(record):
4     nombre = ''
5     dni     = ''
6     edad    = 0
7
8 def copia(pers):
9     return Persona(nombre=pers.nombre[:], dni=pers.dni[:], edad=pers.edad)
10
11 def nada_util(persona1, persona2):
12     persona1.edad = persona1.edad + 1
13     persona3 = persona2
14     persona4 = copia(persona2)
15     persona3.edad = persona3.edad - 1
16     persona4.edad = persona4.edad - 2
17     return persona4
18
19 juan = Persona(nombre='Juan_Paz', dni='12345679Z', edad=19)
20 pedro = Persona(nombre='Pedro_López', dni='23456789D', edad=18)
21 otro = nada_util(juan, pedro)
22 print juan
23 print pedro
24 print otro

```

Haz un diagrama que muestre el estado de la memoria en los siguientes instantes:

1. justo antes de ejecutar la línea 19,
2. justo antes de ejecutar la línea 15 en la invocación de *nada_util* desde la línea 19,
3. al finalizar la ejecución del programa.

► **408** Modifica las rutinas *listado_completo* y *listado_de_nombres* para que los estudiantes aparezcan por orden alfabético. Quizá te convenga definir una función auxiliar que recibe la lista de estudiantes y la ordena alfabéticamente.

► **409** Modifica cuanto consideres necesario para que la lista de estudiantes esté *siempre* ordenada alfabéticamente.

► **410** Diseña un procedimiento que, dada una lista de estudiantes y un grupo (la letra A, B o C), muestre por pantalla un listado con el nombre de los estudiantes de dicho grupo.

► **411** Define una función *esta_aprobado* que devuelva *True* si el alumno ha aprobado la asignatura y *False* en caso contrario.

► **412** Modifica *muestra_acta* para que, además, muestre la calificación numérica (nota del examen) de los alumnos presentados. En los no presentados no debe figurar valor numérico alguno.

► **413** Diseña una función que devuelva el porcentaje de aprobados sobre el total de estudiantes (y no sobre el total de estudiantes que han entregado la práctica).

► **414** Diseña un procedimiento que muestre en pantalla el nombre de todos los estudiantes cuya nota de examen es superior a la media, hayan entregado la práctica o no.

► **415** Diseña un procedimiento que muestre en pantalla el nombre de todos los estudiantes cuya nota de examen es superior a la media y hayan entregado la práctica.

► **416** Diseña una función que reciba una lista de estudiantes y el código de un grupo (la letra A, B o C) y devuelva la nota media en dicho grupo.

► **417** Diseña una función que ordene alfabéticamente la lista de estudiantes por su nombre.

► **418** Diseña una función que ordene la lista de estudiantes por la calificación obtenida en el examen.

► **419** Diseña una función que ordene la lista de estudiantes por la calificación final obtenida. En primer lugar aparecerán las notas más altas y en último lugar los no presentados.

► **420** Deseamos realizar un programa que nos ayude a gestionar nuestra colección de ficheros MP3. Cada fichero MP3 contiene una canción y deseamos almacenar en nuestra base de datos la siguiente información de cada canción:

- título,
- intérprete,
- duración en segundos,
- estilo musical.

Empieza definiendo el tipo *MP3*. Cuando lo tengas, define dos procedimientos:

- *muestra_resumen_mp3* : muestra por pantalla sólo el título y el intérprete de una canción (en una sola línea).
- *muestra_mp3* : muestra por pantalla todos los datos de un MP3, con una línea por cada campo.

A continuación, diseña cuantos procedimientos y funciones consideres pertinentes para implementar un menú con las siguientes acciones:

1. añadir una nueva canción a la base de datos (que será una lista de registros *MP3*),
2. listar todos los estilos de los que tenemos alguna canción (cada estilo debe mostrarse una sola vez en pantalla),
3. listar todas las canciones de un intérprete determinado (en formato resumido, es decir, usando el procedimiento *muestra_resumen_mp3*),
4. listar todas las canciones de un estilo determinado (en formato resumido),
5. listar todas las canciones de la base de datos (en formato completo, es decir, llamando a *muestra_mp3*),
6. eliminar una canción de la base de datos dado el título y el intérprete.

(Nota: Si quieres que el programa sea realmente útil, sería interesante que pudieras salvar la lista de canciones a disco duro; de lo contrario, perderás todos los datos cada vez que salgas del programa. En el próximo tema aprenderemos a guardar datos en disco y a recuperarlos, así que este programa sólo te resultará realmente útil cuando hayas estudiado ese tema.)

► **421** Define una función llamada *fecha_larga* que devuelva la fecha en un formato más verboso. Por ejemplo, el 11/9/2001 aparecerá como «11 de septiembre de 2001».

► **422** Diseña una función *fecha_valida* que devuelva *True* si la fecha es válida y *False* en caso contrario. Para comprobar la validez de una fecha debes verificar que el mes esté comprendido entre 1 y 12 y que el día lo esté entre 1 y el número de días que corresponde al mes. Por ejemplo, la fecha 31/4/2000 no es válida, ya que abril tiene 30 días.

Ten especial cuidado con el mes de febrero: recuerda que tiene 29 o 28 días según sea el año bisiesto o no. Usa, si te conviene, la función definida anteriormente.

► **423** Modifica la función *lee_fecha* para que sólo acepte fechas válidas, es decir, fechas cuyo día sea válido para el mes leído. Puedes utilizar la función *fecha_valida* desarrollada en el ejercicio anterior.

► **424** Haz un programa que use el módulo *fecha* y lea una lista de fechas válidas que mostrará después ordenadas de más antigua a más reciente.

► **425** Diseña una función que devuelva cierto si dos fechas son iguales y falso en caso contrario.

► **426** Diseña una función *anyade_un_dia* que añada un día a una fecha dada. La fecha 7/6/2001, por ejemplo, pasará a ser 8/6/2001 tras invocar al método *anyade_un_dia* sobre ella.

Presta especial atención al último día de cada mes, pues su siguiente día es el primero del mes siguiente. Similar atención requiere el último día del año. Debes tener en cuenta que el día que sigue al 28 de febrero es el 29 del mismo mes o el 1 de marzo dependiendo de si el año es bisiesto o no.

► **427** Diseña una función que calcule el número de días transcurridos entre dos fechas que se proporcionan como parámetro. He aquí un ejemplo de uso:

```
>>> from fecha import Fecha, dias_transcurridos ↵
>>> ayer = Fecha(dia=1, mes=1, anyo=2002) ↵
>>> hoy = Fecha(dia=2, mes=1, anyo=2002) ↵
>>> print dias_transcurridos(hoy, ayer) ↵
1
```

(No tengas en cuenta el salto de fechas producido como consecuencia de la reforma gregoriana del calendario. Si no sabes de qué estamos hablando, consulta el cuadro «¿Cuántos días han pasado... dónde?».)

► **428** Usando la función desarrollada en el ejercicio anterior, implementa un programa que calcule biorritmos. Los biorritmos son una de tantas supercherías populares, como el horóscopo o el tarot. Según sus «estudiosos», los ritmos vitales de la persona son periódicos y se comportan como funciones senoidales (\sin). El *ciclo físico* presenta un periodo de 23 días, el *ciclo emocional*, un periodo de 28 días y el *ciclo intelectual*, de 33 días. Si calculas el seno del número de días transcurridos desde la fecha de nacimiento de un individuo y lo normalizas con el periodo de cada ciclo, obtendrás un valor entre -1 (nivel óptimo) y 1 (nivel pésimo) que indica su estado en cada uno de los tres planos: físico, emocional e intelectual. En el periodo «alto», la persona se encuentra mejor en cada uno de los diferentes aspectos:

- En lo físico: mayor fortaleza, confianza, valor y espíritu positivo.
- En lo emocional: mayor alegría y mejor estado de ánimo.
- En lo intelectual: mejores momentos para tomar decisiones y días más aptos para el estudio.

Y en el periodo «bajo», el estado vital empeora:

- En lo físico: cansancio; conviene no someter el cuerpo a grandes excesos de ningún tipo.
- En lo emocional: falta de ambición y mayores fricciones en nuestras relaciones personales.
- En lo intelectual: mayor distracción, falta de atención, poca creatividad y falta de capacidad de cálculo.

Tu programa pedirá una fecha de nacimiento y proporcionará el valor de cada ciclo a día de hoy, acompañado de un texto que resuma su estado en cada uno de los tres planos.

(Te parecerá ridículo, pero hay infinidad de páginas web dedicadas a este asunto.)

► **429** Modifica la función anterior para que sí tenga en cuenta los 10 días «perdidos» en la reforma gregoriana... en España.

► **430** Diseña una función que devuelva el día de la semana (la cadena 'lunes', o 'martes', etc.) en que cae una fecha cualquiera. (Si sabes en que día cayó una fecha determinada, el número de días transcurridos entre esa y la nueva fecha módulo 7 te permite conocer el día de la semana.)

► **431** Diseña un nuevo tipo de registro: *Fecha_con_hora*. Además del día, mes y año, una variable de tipo *Fecha_con_hora* almacena la hora (un número entre 0 y 23) y los minutos (un número entre 0 y 59).

Diseña a continuación funciones que permitan:

- Leer un dato del tipo *Fecha_con_hora* por teclado.
- Mostrar un dato del tipo *Fecha_con_hora* en el formato que ilustramos con este ejemplo: las siete y media de la tarde del 11 de septiembre de 2001 se muestran como 19:30 11/9/2001.
- Mostrar un dato del tipo *Fecha_con_hora* en el formato que ilustramos con este ejemplo: las siete y media de la tarde del 11 de septiembre de 2001 se muestran como 7:30 pm 11/9/2001 y las siete y media de la mañana del mismo día como 7:30 am 11/9/2001.
- Determinar si una *Fecha_con_hora* ocurrió antes que otra.
- Calcular los minutos transcurridos entre dos datos de tipo *Fecha_con_hora*.

► **432** Diseña una función que dado un registro de tipo *Persona* (con fecha de nacimiento) y la fecha de hoy, devuelva la edad (en años) de la persona.

► **433** Diseña un registro denominado *Periodo*. Un periodo consta de dos fechas donde la primera es anterior o igual a la segunda. Diseña entonces:

- a) Un procedimiento *muestra_periodo* que muestre las dos fechas (en formato breve) separadas entre sí por un guión.
- b) Una función que devuelva el número de días comprendidos en el periodo (incluyendo ambos extremos).
- c) Una función que reciba un periodo y una fecha y devuelva cierto si la fecha está comprendida en el periodo y falso en caso contrario.
- d) Una función que reciba dos periodos y devuelva cierto si ambos se solapan (tienen al menos un día en común).

► **434** Define tú mismo las funciones *lee_pelicula*, *contiene_pelicula_con_titulo*, *alta_pelicula* y *baja_pelicula*.

► **435** Detecta posibles fuentes de ineficiencia (llamadas a función repetidas) en el fragmento de programa anterior y corrígelas.

- **436** Añade nueva funcionalidad al programa: una opción que permita devolver una película alquilada. Diseña para ello un procedimiento *devolver_pelicula*. A continuación, añade una opción al menú para devolver una película. Las acciones asociadas son:
- pedir el nombre de la película;
 - si no existe una película con ese título, dar el aviso pertinente con un mensaje por pantalla y no hacer nada más;
 - si existe la película pero no estaba alquilada, avisar al usuario y no hacer nada más;
 - y si existe la película y estaba alquilada, «marcarla» como disponible (poner a *None* su campo *alquilada*).
- **437** Modifica la porción del programa que da de baja a un socio o a una película para que no se permita dar de baja una película que está actualmente alquilada ni a un socio que tiene alguna película en alquiler. Te convendrá disponer de una función que compruebe si una película está disponible y, por tanto, se puede dar de baja y otra que compruebe si un socio tiene alguna película en alquiler actualmente. Modifica las acciones asociadas a las respectivas opciones del menú para que den los avisos pertinentes en caso de que no sea posible dar de baja a un socio o una película.
- **438** Diseña una función *listado_completo_por_genero* que muestre los títulos de todas las películas del videoclub del género que se indique, pero indicando al lado de cada título si la correspondiente película está alquilada o disponible. Y, ya puestos, haz que el listado de películas aparezca en pantalla ordenado alfabéticamente por su título.
- **439** Implementa la nueva función de devolución de películas. Ten en cuenta que necesitarás dos datos: el título de la película y el DNI del socio.
- **440** Modifica la definición de *Pelicula* para añadir los nuevos campos. Modifica a continuación *lee_pelicula* para que pida también el valor de *dias_permitidos*.
- **441** Modifica el método de devolución de películas para que tenga en cuenta la fecha de alquiler y la fecha de devolución. El método devolverá el número de días de retraso. Si no hay retraso, dicho valor será cero. (Usa la función *dias_transcurridos* del módulo *fecha* para calcular el número de días transcurridos desde una fecha determinada.)
Modifica las acciones asociadas a la opción de menú de devolución de películas para que tenga en cuenta el valor devuelto por *devolver_pelicula* y muestre por pantalla el número de días de retraso (si es el caso).
- **442** Modifica el método *listado_completo_por_genero* (ejercicio 438) para que los títulos no aparezcan repetidos en el caso de que dispongamos de más de un ejemplar de una película. Al lado del título aparecerá el mensaje «disponible» si hay al menos un ejemplar disponible y «no disponible» si *todos* los ejemplares están alquilados.
- **443** Modifica el programa para permitir que una película sea clasificada en diferentes géneros. (El atributo *genero* será una lista de cadenas, y no una simple cadena.)
- **444** Modifica la aplicación para permitir reservar películas a socios. Cuando no se disponga de ningún ejemplar libre de una película, los socios podrán solicitar una reserva.
¡Ojo!, la reserva se hace sobre una película, no sobre un ejemplar, es decir, la lista de espera de «Matrix» permite a un socio alquilar el primer ejemplar de «Matrix» que quede disponible. Si hay, por ejemplo, dos socios con un mismo título reservado, sólo podrá alquilarse a otros socios un ejemplar de la película cuando haya tres o más ejemplares libres.
- **445** Modifica el programa del ejercicio anterior para que las reservas caduquen automáticamente a los dos días. Es decir, si el socio no ha alquilado la película a los dos días de estar disponible, su reserva expira.
- **446** Modifica el programa para que registre el número de veces que se ha alquilado cada película. Una opción de menú permitirá mostrar la lista de las 10 películas más alquiladas hasta el momento.
- **447** Modifica el programa para que registre todas las películas que ha alquilado cada socio a lo largo de su vida.
Añade una opción al menú de la aplicación que permita consultar el género (o géneros) favorito(s) de un cliente a partir de su historial de alquileres.
- **448** Añade al programa una opción de menú para aconsejar al cliente. Basándose en su historial de alquileres, el programa determinará su género (o géneros) favorito(s) y mostrará un listado con las películas de dicho(s) género(s) disponibles para alquiler en ese instante (ten en cuenta que las películas disponibles sobre las que hay lista de espera no siempre se pueden considerar realmente disponibles).
- **449** Nos gustaría retomar el programa de gestión de MP3 que desarrollamos en un ejercicio anterior. Nos gustaría introducir el concepto de «álbum». Cada álbum tiene un título, un(os) intérprete(s) y una lista de canciones (archivos MP3). Modifica el programa para que gestione álbumes. Deberás permitir que el usuario dé de alta y baja álbumes, así como que obtenga listados completos de los álbumes disponibles, listados ordenados por intérpretes, búsquedas de canciones en la base de datos, etc.

► **450** Deseamos gestionar una biblioteca. La biblioteca contiene libros que los socios pueden tomar prestados un número de días. De cada libro nos interesa, al menos, su título, autor y año de edición. De cada socio mantenemos su DNI, su nombre y su teléfono. Un socio puede tomar prestados tres libros. Si un libro tarda más de 10 días en ser devuelto, el socio no podrá sacar nuevos libros durante un período de tiempo: tres días de penalización por cada día de retraso.

Diseña un programa que permita dar de alta y baja libros y socios y llevar control de los préstamos y devoluciones de los libros. Cuando un socio sea penalizado, el programa indicará por pantalla hasta qué fecha está penalizado e impedirá que efectúe nuevos préstamos hasta entonces.

► **451** Diseña un programa que cuente el número de caracteres de un fichero de texto, incluyendo los saltos de línea. (El nombre del fichero se pide al usuario por teclado.)

► **452** Haz un programa que, dada una palabra y un nombre de fichero, diga si la palabra aparece o no en el fichero. (El nombre del fichero y la palabra se pedirán al usuario por teclado.)

► **453** Haz un programa que, dado un nombre de fichero, muestre cada una de sus líneas precedida por su número de línea. (El nombre del fichero se pedirá al usuario por teclado.)

► **454** Haz una *función* que, dadas la ruta de un fichero y una palabra, devuelva una lista con las líneas que contienen a dicha palabra.

Diseña a continuación un programa que lea el nombre de un fichero y tantas palabras como el usuario desee (utiliza un bucle que pregunte al usuario si desea seguir introduciendo palabras). Para cada palabra, el programa mostrará las líneas que contienen dicha palabra en el fichero.

► **455** Haz un programa que muestre por pantalla la línea más larga de un fichero. Si hay más de una línea con la longitud de la más larga, el programa mostrará únicamente la primera de ellas. (El nombre del fichero se pedirá al usuario por teclado.)

► **456** Haz un programa que muestre por pantalla todas las líneas más largas de un fichero. (El nombre del fichero se pedirá al usuario por teclado.) ¿Eres capaz de hacer que el programa lea una sola vez el fichero?

► **457** La orden `head` («cabeza», en inglés) de Unix muestra las 10 primeras líneas de un fichero. Haz un programa `head.py` que muestre por pantalla las 10 primeras líneas de un fichero. (El nombre del fichero se pedirá al usuario por teclado.)

► **458** En realidad, la orden `head` de Unix muestra las n primeras líneas de un fichero, donde n es un número suministrado por el usuario. Modifica `head.py` para que también pida el valor de n y muestre por pantalla las n primeras líneas del fichero.

► **459** La orden `tail` («cola», en inglés) de Unix muestra las 10 últimas líneas de un fichero. Haz un programa `tail.py` que muestre por pantalla las 10 *últimas* líneas de un fichero. (El nombre del fichero se pide al usuario por teclado.) ¿Eres capaz de hacer que tu programa lea una sola vez el fichero? Pista: usa una lista de cadenas que almacene las 10 últimas cadenas que has visto en cada instante.

► **460** Modifica `tail.py` para que pida un valor n y muestre las n últimas líneas del fichero.

► **461** El fichero `/etc/passwd` de los sistemas Unix contiene información acerca de los usuarios del sistema. Cada línea del fichero contiene datos sobre un usuario. He aquí una línea de ejemplo:

```
al55555:x:1000:2000:Pedro Pérez:/home/al55555:/bin/bash
```

En la línea aparecen varios campos separados por dos puntos (:). El primer campo es el nombre clave del usuario; el segundo *era* la contraseña cifrada (por razones de seguridad, ya no está en `/etc/passwd`); el tercero es su número de usuario (cada usuario tiene un número diferente); el cuarto es su número de grupo (en la UJI, cada titulación tiene un número de grupo); el quinto es el nombre real del usuario; el sexto es la ruta de su directorio principal; y el séptimo es el intérprete de órdenes.

Haz un programa que muestre el nombre de todos los usuarios reales del sistema.

(Nota: recuerda que el método `split` puede serte de gran ayuda.)

► **462** Haz un programa que pida el nombre clave de un usuario y nos diga su nombre de usuario real utilizando `/etc/passwd`. El programa no debe leer todo el fichero a menos que sea necesario: tan pronto encuentre la información solicitada, debe dejar de leer líneas del fichero.

► **463** El fichero `/etc/group` contiene una línea por cada grupo de usuarios del sistema. He aquí una línea de ejemplo:

```
gestion:x:2000:
```

Al igual que en `/etc/passwd`, los diferentes campos aparecen separados por dos puntos. El primer campo es el nombre del grupo; el segundo no se usa; y el tercero es el número de grupo (cada grupo tiene un número diferente).

Haz un programa que solicite al usuario un nombre de grupo. Tras consultar `/etc/group`, el programa listará el nombre real de todos los usuarios de dicho grupo relacionados en el fichero `/etc/passwd`.

► **464** El comando `wc` (por «word count», es decir, «conteo de palabras») de Unix cuenta el número de bytes, el número de palabras y el número de líneas de un fichero. Implementa un comando `wc.py` que pida por teclado el nombre de un fichero y muestre por pantalla esos tres datos acerca de él.

- **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.

- **477** Modifica la aplicación de gestión de estudiantes del capítulo anterior para que recuerde todos los datos entre ejecución y ejecución. (Puedes inspirarte en la segunda versión de la agenda.)
- **478** Modifica la aplicación de gestión del videoclub del capítulo anterior para que recuerde todos los datos entre ejecución y ejecución. Mantén dos ficheros distintos: uno para las películas y otro para los socios.
- **479** Diseña un programa que lea un fichero de texto en formato HTML y genere otro en el que se sustituyan todos los fragmentos de texto resaltados en negrita por el mismo texto resaltado en cursiva.
- **480** Las cabeceras (títulos de capítulos, secciones, subsecciones, etc.) de una página *web* se marcan encerrándolas entre `<Hn>` y `</Hn>`, donde n es un número entre 1 y 6 (la cabecera principal o de nivel 1 se encierra entre `<H1>` y `</H1>`). Escribe un programa para cada una de estas tareas sobre un fichero HTML:
 - mostrar únicamente el texto de las cabeceras de nivel 1;
 - mostrar el texto de todas las cabeceras, pero con sangrado, de modo que el texto de las cabeceras de nivel n aparezca dos espacios más a la derecha que el de las cabeceras de nivel $n - 1$.

Un ejemplo de uso del segundo programa te ayudará a entender lo que se pide. Para el siguiente fichero HTML,

```

1 <HTML>
2 <BODY>
3 <H1>Un_titular</H1>
4 <P>Texto_en_un_párrafo.
5 <P>Otro_párrafo.
6 <H1>Otro_titular</H1>
7 <H2>Un_subtítulo</H2>
8 <P>Y_su_texto.
9 <H3>Un_subsubtítulo</H3>
10 <H2>Otro_subtítulo</H2>
11 <P>Y_el_suyo
12 </BODY>
13 </HTML>
```

el programa mostrará por pantalla:

```

Un titular
Otro titular
    Un subtítulo
        Un subsubtítulo
    Otro subtítulo
```

- **481** Añade una opción a la agenda desarrollada en el apartado anterior para que genere un fichero **agenda.html** con un volcado de la agenda que podemos visualizar en un navegador *web*. El listado aparecerá ordenado alfabéticamente (por apellido), con una sección por cada letra del alfabeto y una línea por entrada. El apellido de cada persona aparecerá destacado en negrita.
- **482** Modifica el programa **agenda2.py** para que asuma un formato de **agenda.txt** similar al **/etc/passwd**. Cada línea contiene una entrada y cada entrada consta de 3 o más campos separados por dos puntos. El primer campo es el nombre, el segundo es el apellido y el tercero y posteriores corresponden a diferentes teléfonos de esa persona.
- **483** Un programa es, en el fondo, un fichero de texto con formato, aunque bastante complicado, por regla general. Cuando ejecuta un programa el intérprete está, valga la redundancia, interpretando su significado paso a paso. Vamos a diseñar nosotros mismos un intérprete para un pequeño lenguaje de programación. El lenguaje sólo tiene tres variables llamadas *A*, *B* y *C*. Puedes asignar un valor a una variable con sentencias como las de este programa:

```

1 asigna_A_suma_3_y_7
2 asigna_B_resta_A_y_2
3 asigna_C_producto_A_y_B
4 asigna_A_division_A_y_10
```

Si interpretas ese programa, *A* acaba valiendo 1, *B* acaba valiendo 8 y *C* acaba valiendo 80. La otra sentencia del lenguaje permite mostrar por pantalla el valor de una variable. Si añades al anterior programa estas otras sentencias:

```

1 muestra_A
2 muestra_B
```

obtendrás en pantalla una línea con el valor 1 y otra con el valor 8.

Diseña un programa que pida el nombre de un fichero de texto que contiene sentencias de nuestro lenguaje y muestre por pantalla el resultado de su ejecución. Si el programa encuentra una sentencia incorrectamente escrita (por ejemplo `muestrame A`), se detendrá mostrando el número de línea en la que encontró el error.

► **484** Enriquece el intérprete del ejercicio anterior para que entienda la orden *si valor condición valor entonces línea número*. En ella, *valor* puede ser un número o una variable y *condición* puede ser la palabra *igual* o la palabra *distinto*. La sentencia se interpreta como que si es cierta la condición, la siguiente línea a ejecutar es la que tiene el número *número*.

Si tu programa Python interpreta este programa:

```
1 asigna_A_suma_0_y_1
2 asigna_B_suma_0_y_1
3 muestra_B
4 asigna_B_producto_2_y_B
5 asigna_A_suma_A_y_1
6 si_A_distinto_8_entonces_linea_3
```

en pantalla aparecerá

```
1
2
4
8
16
32
64
```