6.6.2. Aproximación de la exponencial de un número real

Vamos a desarrollar una función que calcule el valor de e^a, siendo a un número real, con una restricción: no podemos utilizar el operador de exponenciación **.

Si a fuese un número natural, sería fácil efectuar el cálculo:

exponencial_8.py — exponencial.py

```
from math import e

def exponencial(a):
    exp = 1
    for i in range(a):
        exp *= e
    return exp
```

EJERCICIOS

▶ 359 ¿Y si a pudiera tomar valores enteros negativos? Diseña una función *exponencial* que trate también ese caso. (Recuerda que $e^{-a} = 1/e^a$.)

Pero siendo a un número real (bueno, un flotante), no nos vale esa aproximación. Refrescando conocimientos matemáticos, vemos que podemos calcular el valor de e^a para a real con la siguiente fórmula:

$$e^a = 1 + a + \frac{a^2}{2} + \frac{a^3}{3!} + \frac{a^4}{4!} + \cdots + \frac{a^k}{k!} + \cdots = \sum_{n=0}^{\infty} \frac{a^n}{n!}.$$

La fórmula tiene un número infinito de sumandos, así que no la podemos codificar en Python. Haremos una cosa: diseñaremos una función que aproxime el valor de e^a con tantos sumandos como nos indique el usuario.

Vamos con una primera versión:

exponencial_9.py — exponencial.py

```
def exponencial(a, n):
    sumatorio = 0.0
    for k in range(n):
        sumatorio += a**k / (k!)
    return sumatorio
```

Mmmm. Mal. Por una parte, nos han prohibido usar el operador **, así que tendremos que efectuar el correspondiente cálculo de otro modo. Recuerda que

$$a^k = \prod_{i=1}^{k} a.$$

exponencial_10.py — exponencial.py

```
def exponencial(a, n):
    sumatorio = 0.0
    for k in range(n):
        # Cálculo de a^k.
        numerador = 1.0
        for i in range(1, k+1):
            numerador *= a
        # Adición de nuevo sumando al sumatorio.
        sumatorio += numerador / k!
    return sumatorio
```

Y por otra parte, no hay operador factorial en Python. Tenemos que calcular el factorial explícitamente. Recuerda que

$$k! = \prod_{i=1}^{k} i.$$

Corregimos el programa anterior:

exponencial_11.py — exponencial.py

```
def exponencial(a, n):
    sumatorio = 0.0
    for k in range(n):
        # Cálculo de a^k.
        numerador = 1.0
        for i in range(1, k+1):
            numerador *= a
        # Cálculo de k!.
        denominador = 1.0
        for i in range(1, k+1):
            denominador *= i
        # Adición de nuevo sumando al sumatorio.
        sumatorio += numerador / denominador
    return sumatorio
```

Y ya está. La verdad es que no queda muy legible. Analiza esta otra versión:

exponencial_12.py — exponencial.py

```
def elevado(a, k):
    productorio = 1.0
    for i in range(1, k+1):
        productorio *= a
    return productorio

def factorial(k):
    productorio = 1.0
    for i in range(1, k+1):
        productorio *= i
    return productorio

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

Esta versión es mucho más elegante que la anterior, e igual de correcta. Al haber separado el cálculo de la exponenciación y del factorial en sendas funciones hemos conseguido que la función *exponencial* sea mucho más legible.

EJERCICIOS

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

exponencial_13.py — exponencial.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

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!}$$

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!}$$

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

El método de cálculo utilizado en la función *exponencial* es ineficiente: el término *elevado*(*a*, *k*) / *factorial*(*k*) resulta costoso de calcular. Imagina que nos piden calcular *exponencial*(*a*, 8). Se producen la siguientes llamadas a *elevado* y *factorial*:

- *elevado*(*a*, 0) y *factorial*(0).
- *elevado*(*a*, 1) y *factorial*(1).
- *elevado*(*a*, 2) y *factorial*(2).
- *elevado*(*a*, 3) y *factorial*(3).
- *elevado*(*a*, 4) y *factorial*(4).
- *elevado*(*a*, 5) y *factorial*(5).
- *elevado*(*a*, 6) y *factorial*(6).
- *elevado*(*a*, 7) y *factorial*(7).

Estas llamadas esconden una repetición de cálculos que resulta perniciosa para la velocidad de ejecución del cálculo. Cada llamada a una de esas rutinas supone iterar un bucle, cuando resulta innecesario si aplicamos un poco de ingenio. Fíjate en que se cumplen estas dos relaciones:

$$elevado(a, n) = a * elevado(a, n-1), \qquad factorial(n) = n * factorial(n-1),$$

para todo n mayor que 0. Si n vale 0, tanto *elevado*(*a*, *n*) como *factorial*(*n*) valen 1. Este programa te muestra el valor de *elevado*(2, *n*) para n entre 0 y 7:

elevado_rapido.py

```
a = 2
valor = 1
print 'elevado(%d,␣%d)␣=␣%d' % (a, 0, valor)
for n in range(1, 8):
    valor = a * valor
    print 'elevado(%d,␣%d)␣=␣%d' % (a, n, valor)
```

```
elevado(2, 0) = 1
elevado(2, 1) = 2
elevado(2, 2) = 4
elevado(2, 3) = 8
elevado(2, 4) = 16
elevado(2, 5) = 32
elevado(2, 6) = 64
elevado(2, 7) = 128
```

EJERCICIOS

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

Explotemos esta forma de calcular esa serie de valores en el cómputo de *exponencial*:

exponencial_14.py

```
exponencial.py
1 def exponencial(a, n):
2     numerador = 1
3     denominador = 1
4     sumatorio = 1.0
5     for k in range(1, n):
6         numerador = a * numerador
7         denominador = k * denominador
8         sumatorio += numerador / denominador
9     return sumatorio
```

EJERCICIOS

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

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

Cuando *n* vale 0, tenemos:

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

Resolvamos ahora un problema ligeramente diferente: vamos a aproximar e^a con tantos términos como sea preciso hasta que el último término considerado sea menor o igual que un valor ϵ dado. Lo desarrollaremos usando, de nuevo, las funciones *elevado* y *factorial*. (Enseguida te pediremos que mejores el programa con las últimas ideas presentadas.) No resulta apropiado ahora utilizar un bucle **for-in**, pues no sabemos cuántas iteraciones habrá que dar hasta llegar a un $a^k/k!$ menor o igual que ϵ. Utilizaremos un bucle **while**:

```
exponencial.py
1  def elevado(a, k):
2      productorio = 1.0
3      for i in range(k):
4          productorio *= a
5      return productorio
6
7  def factorial(n):
8      productorio = 1.0
9      for i in range(1, n+1):
10         productorio *= i
11     return productorio
12
13 def exponencial2(a, epsilon):
14     sumatorio = 0.0
15     k = 0
16     termino = elevado(a, k) / factorial(k)
17     while termino > epsilon:
```

```
18          sumatorio += termino
19          k += 1
20          termino = elevado(a, k) / factorial(k)
21      return sumatorio
```

EJERCICIOS

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