Práctica 4

April 25, 2021

1 Práctica 4

1.1 Herramientas Computacionales

2 Descripción del ejercicio

El objetivo de este ejercicio es utilizar un diccionario como una forma simple de caché de datos.

Calcular el factorial para un número muy grande puede llevar algún tiempo. Por ejemplo, calcular el factorial de 150000 puede llevar varios segundos. Podemos verificar esto usando un temporizador.

El siguiente programa ejecuta varios cálculos factoriales en números grandes e imprime el tiempo necesario para cada uno:

[3]:
```
from timeit import default_timer
```

[4]:
```
def timer(func):

    def inner(value):
        print('calling ', func.__name__, 'with', value)
        start = default_timer()
        func(value)
        end = default_timer()
        print('returned from ', func.__name__, 'it took', int(end - start),␣
↪'seconds')

    return inner
```

2.1 Función factorial

[5]:
```
@timer
def factorial(num):
    if num == 0:
        return 1

    else:
        factorial_value = 1

        for i in range(1, num + 1):
```

```
            factorial_value = factorial_value * i
        return factorial_value
```

2.2 Prueba con 80000

[6]:
```
print(factorial(80000))
```

```
calling  factorial with 80000
returned from  factorial it took 3 seconds
None
```

2.3 Prueba con 120000

[7]:
```
print(factorial(120000))
```

```
calling  factorial with 120000
returned from  factorial it took 7 seconds
None
```

2.4 Prueba con 150000

[8]:
```
print(factorial(150000))
```

```
calling  factorial with 150000
returned from  factorial it took 11 seconds
None
```

Como puede verse a partir de esto, en esta ejecución en particular, calcular el factorial de 150000 tomó 9 s, mientras que el factorial de 80000 tomó 2 s, etc.

En este caso particular, hemos decidido volver a ejecutar estos cálculos para que realmente hayamos calculado el factorial de 150000, 80000 y 120000 al menos dos veces.

[9]:
```
print(factorial(80000))
```

```
calling  factorial with 80000
returned from  factorial it took 2 seconds
None
```

[10]:
```
print(factorial(120000))
```

```
calling  factorial with 120000
returned from  factorial it took 7 seconds
None
```

[11]:
```
print(factorial(150000))
```

```
calling  factorial with 150000
returned from  factorial it took 11 seconds
None
```

La idea de un caché es que se puede usar para guardar cálculos anteriores y reutilizarlos si es apropiado en lugar de tener que realizar el mismo cálculo varias veces. El uso de una caché puede mejorar en gran medida el rendimiento de los sistemas en los que se producen estos cálculos repetidos.

Hay muchas bibliotecas comerciales de almacenamiento en caché disponibles para una amplia variedad de lenguajes, incluido Python. Sin embargo, en su esencia, todos son algo así como un diccionario; es decir, hay una llave `key` que suele ser una combinación de la operación invocada y los valores `values` de los parámetros utilizados. A su vez, el elemento de valor `value` es el resultado del cálculo.

Estos cachés también suelen tener políticas de desalojo para que no se vuelvan demasiado grandes; Por lo general, estas políticas de desalojo se pueden especificar para que coincidan con la forma en que se utiliza la caché. Una política de desalojo común es la política de uso menos reciente. Cuando se utiliza esta política, una vez que el tamaño de la caché alcanza un límite predeterminado, se elimina el valor de uso menos reciente, etc.

Para este ejercicio, debe implementar un mecanismo de almacenamiento en caché simple utilizando un diccionario (pero sin una política de desalojo).

La caché debe usar el parámetro pasado a la función `factorial()` como clave y devolver el valor almacenado si hay uno presente.

La lógica para esto suele ser:

- Revise en la caché para ver si la clave está presente
- Si está, devuelva el valor
- Si no realiza el cálculo
- Almacene el resultado calculado para uso futuro
- Devuelve el valor

Tenga en cuenta que la función `factorial()` es exactamente una función; Deberá pensar en usar una variable global para mantener el caché.

Una vez que se usa la caché con la función `factorial()`, cada invocación posterior de la función que usa un valor anterior debería regresar casi de inmediato. Esto se muestra en la salida de muestra anterior, donde las llamadas de método posteriores regresan en menos de un segundo

3 Implementación del ejercicio

- ***Del notebook `Functions.ipybn` o en el siguiente enlace investigue que es una variable global y describalo a continuación (1 punto)***

[ ]:

- ***En la siguiente celda se define un diccionario que se llama `factorials`, con las siguientes llaves y valores, `0:1,5:120` y `10:3628800` observe que la llave es el número al que le deseamos calcular el factorial y el valor es el factorial del número. Ejecute la celda y argumente por qué es una variable global (1 punto)***

[12]:
```
factorials = {0:1, 5:120, 10:3628800}
```

[13]:
```
factorials[0]
```

[13]: 1

- *Implemente una función que se llame* `check_factorial(n, factorials)`*, donde* `n` *es un numero natural. La función debe de revisar si el número* n *está en el diccinario* `factorials` *y si lo está debe de regresar una tupla con* n *y el factorial de* n*, de lo contrario si* n *no está en el diccionario debe de regresar en una tupla el número previo a* n *que si esté en el diccionario y su factorial (3 puntos)*
- *Si su implementación es correcta debe aparecer lo siguiente:*

```
check_factorial(0,factorials)
(0, 1)

check_factorial(2,factorials)
(0, 1)

check_factorial(5,factorials)
(5, 120)

check_factorial(8,factorials)
(5, 120)

check_factorial(10,factorials)
(10, 3628800)

check_factorial(15,factorials)
(10, 3628800)
```

[ ]:
```

```

- *Modifique la función Section ?? cuyo nombre es* `factorial(n)` *para que está tome como argumentos* `n` *el número al que se desea calcular. Use la función* `check_factorial(n, factors)`*,dentro de* `factorial(n)`*, para obtener factorial de algún numero previo y a partir de este calcule el factotial del* `n` *(5 puntos)*

[ ]:
```

```

- *Ejemplo. Supogamos que en* `factorials` *ya tiene almacenado el factorial de* 80000*, y usted desea calcular el factorial de* 120000*, entonces la función* `factorial` *debe se usar el factorial de* 80000 *que se encuentra en el diccionario* `factorials` *para calular el factorial de* 120000

Respuesta:

[22]:
```
@timer
def factorial(num):
    """La función nos devolvera el tiempo en segundos en que el programa se␣
 ↪tarda en decolver el radical
```

```
    del número ingresado"""

    n, factorial_value = check_factorial(num,factorials)

    if num == n:
        return factorial_value

    else:

        for a in range(n, num + 1):
            factorial_value = factorial_value * a
        factorialss[num] = factorial_value
        return factorial_value
```

- *Ejecute la siguientes celdas y observe que el resultado debe de ser el siguiente:

```
print(factorial(10))
calling  factorial with 10
returned from  factorial it took 0 seconds
None

print(factorial(80000))
calling  factorial with 80000
returned from  factorial it took 2 seconds
None

print(factorial(120000))
calling  factorial with 120000
returned from  factorial it took 3 seconds
None
```

- ***Observe que los tiempo de ejecución son menores que el caso anterior, explíque porque (1 punto)***

[ ]: