Curso de Introducción al Pensamiento Computacional con Python

Artículo

Representación de flotantes

David Aroesti 11 de Diciembre de 2019

La mayoría del tiempo los números flotantes (tipo `float`) son una muy buena aproximación de los números que queremos calcular con nuestras computadoras. Sin embargo, "la mayoría del tiempo" no significa todo el tiempo, y cuando no se comportan de esta manera puede tener consecuencias inesperadas.

Por ejemplo, trata de correr el siguiente código:

```
x = 0.0
for i in range(10):
    x += 0.1

if x == 1.0:
    print(f'x = {x}')
else:
    print(f'x != {x}')
```

Es probable que te hayas sorprendido con el resultado. La mayoría de nosotros esperaríamos que imprimiera `1.0` en vez de `0.999999999999`. ¿Qué es lo que pasó?.

Para entender qué es lo que pasó tenemos que entender que es lo que pasa en la computadora cuando realizamos cómputos con números flotantes. Y para eso necesitamos entender números binarios.

Cuando aprendiste a contar, lo que en realidad aprendiste es una técnica combinatoria para manipular los siguientes símbolos que le llamamos números: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

La forma en la que funciona esta técnica es asignando el número 10 a la 0 al número de la extrema derecha, 10 a la 1 al siguiente, 10 a la 2 al siguiente y así sucesivamente. De tal manera que el número 525 es simplemente la representación de `(5 * 100) + (2 * 10) + (5 * 1)`.

Esto nos dice que el número de números que podemos representar depende de cuanto espacio tengamos. Si tenemos un espacio de 3, podemos representar 1,000 números (10 elevado a la 3) o la secuencia del 0 al 999. Si tenemos 4, podemos representar 10,000 (10 elevado a la 4) o la secuencia del 0 al 9,999. De manera general podemos decir que con una secuencia de tamaño n, podemos representar 10 elevado a la n números.

Los números binarios funcionan de la misma manera (de hecho cualquier número en cualquier base, por ejemplo, octales o hexadecimales). La única diferencia es cuántos símbolos tenemos para representar. En binario nada más tenemos 0, 1;
en hexadecimal tenemos 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f.

De esta manera podemos decir que el número de la extrema derecha es `cantidad_de_simbolos**0`, `cantidad_de_simbolos**1`, `cantidad_de_simbolos**2`, etc. Por lo que en binario, que nada más tenemos 2 símbolos, decimos `2**0`, `2**1`, `2**2`, etc. Por ejemplo el número binario 101 es la representación de `(1 * 4) + (0 * 2) + (1 * 1)`, es decir 5.

Esta representación nos permite trabajar con todos los números positivos enteros dentro del computador, pero ¿Qué hacemos con los negativos y los racionales?.

El caso de los números negativos es sencillo: simplemente agregamos un bit adicional que representa el signo y la añadimos en la extrema izquierda. Por lo que el número `0`101 sería +5 y el número `1`101 sería -5.

El caso de los racionales es más complejo. En la mayoría de los lenguajes de programación modernos los racionales utilizan una implementación llamada punto flotante. ¿Cómo funciona esta representación?.

Antes de pasar a binario, vamos a pretender que estamos trabajando con una computadora basada en decimales. Un número flotante lo representaríamos con un par de enteros: los dígitos significativos y el exponente. Por ejemplo, el número 2.345 se representaría como `(2345 * 10**-3)` o `(2345, -3)`.

El número de dígitos significativos determinan la precisión con la que podemos representar número. Por ejemplo si nada más tuviéramos dos dígitos significativos el número 2.345 no se podría representar de manera exacta y tendríamos que convertirlo a una aproximación, en este caso 2.3.

Ahora pasemos a la verdadera representación interna de la computadora, que es en binario. ¿Cómo representarías el número 5/8 o 0.625? Lo primero que tenemos que saber es que 5/8 es en realidad el número `5 * 2**-3`. Por lo que podríamos decir (101, -11) (recuerda que el número 5 es 101 en binario y el 3 es 11).

Regresemos a nuestro problema inicial: ¿Cómo representaremos 1/10 (que escribimos en Python cómo 0.1)? Lo mejor que podemos hacer con cuatro dígitos significativos es (0011, -101) que es equivalente a 3/32 (0.09375). ¿Qué tal si tuviéramos cinco dígitos significativos? La mejor representación sería (11001, -1000) que es equivalente a 25/256 (0.09765625). ¿Cuántos dígitos significativos necesitamos entonces? Un número infinito. No existe ningún número que cumpla con la siguiente ecuación: `sim * 2**-exp`.

En la mayoría de las implementaciones de Python tenemos 53 bits de precisión para números

1100110011001100110011001100110011001100110011001100110011001 que es equivalente al número decimal: 0.1000000000000000055511151231257827021181583404541015625

Muy cercano a 1/10 pero no exactamente 1/10. Ahora ya sabemos la razón de esa respuesta tan extraña. Hay muy pocas situaciones en la que 1.0 es aceptable, pero 0.9999999999999999 no. Pero ¿Cuál es la moraleja de esta historia?

Hasta ahora hemos verificado igualdad con el operador `==`. Sin embargo, cuando estamos trabajando con flotantes es mejor asegurarnos que los números sean aproximados en vez de idénticos. Por ejemplo `x < 1.0 and x > 0.99999`.