numeros-y-matematica

January 28, 2020

1 Numeros y Matematica

No necesitamos ser matematicos para programar. La verdad es que pocos programadores necesitan saber mas que algebra básica. Claro que la cantidad de matematica que necesitas saber esta directamente relacionada al tipo de aplicación en la cual estas trabajando. En general, el nivel de matematica requerido para trabajar como programador es menor al nivel establecido mediante expectativas de terceros.

Aunque la matematica y la programación no estan tan corelacionadas como alguna gente puede creer, los numeros constituyen parte integral de cualquier lenguaje de programación; y Python no es la excepción.

En esta sección, aprenderemos a: * Trabajar con los tres tipos de numeros en Python: * integer o numero entero; * floating-point o punto-flotante (numero real); * numeros complejos * Redondear numeros * Formatear y mostrar numeros en strings

1.1 Numeros enteros y de punto-flotante

Python tiene tres tipos de numeros: * integer o numero entero; * floating-point o punto-flotante; * numeros complejos

1.1.1 Numeros enteros

Un numero entero no tiene decimales. Por ejemplo, `1` es un numero entero, pero `1.0` no lo es. El nombre oficial para numeros enteros como tipo de datos es `int`, lo cual podemos obserbar con la funcion `type()`.

[1]:
```
type(1)
```

[1]:
```
int
```

[2]:
```
# podemos crear un numero escribiendo el numero sin las citas o con int()
int('1')
```

[2]:
```
1
```

Un literal de numero entero (integer literal) es un numero entero cuyo valor esta explicitamente escrito en el codigo, de la misma manera que un literal de cadena de caracteres es un string que esta explicitamente escrito en el codigo. Por ejemplo, `1` es un literal de numero entero, pero `int('1')` no lo es.

Podemos escribir un literal de numero entero en dos formas:

[4]:
```
100000000
```

[4]:
```
100000000
```

[5]:
```
1_000_000  # en vez de comas, utilizamos _
```

[5]:
```
1000000
```

1.1.2 Numeros de punto flotante

Un numero de punto flotante es un numero con un punto decimal, por ejemplo `1.0` o `-2.5`. El nombre oficial de un numero de punto flotante como tipo de dato es `float`.

[6]:
```
type(1.0)
```

[6]:
```
float
```

[7]:
```
# podemos crear un numero escribiendo el numero sin las citas o con int()
float('1.0')
```

[7]:
```
1.0
```

Al igual que los numeros enteros, un literal de punto flotante es un numero con punto flotante cuyo valor esta explicitamente escrito en el codigo, por ejemplo `1.5` es un literal de punto flotante, pero `float('1.5')` no lo es.

Podemos crear literales de punt oflotante en tres distintas formas:

[8]:
```
1000000.0
```

[8]:
```
1000000.0
```

[9]:
```
1_000_000.0
```

[9]:
```
1000000.0
```

[10]:
```
1e6
```

[10]:
```
1000000.0
```

Las dos primeras formas son similares a como se crean los numeros enteros, sin embargo la tercera y ultima forma es util para numeros muy grandes. Este método es conocido como **E-notation** o **notacion exponencial**. Python agarra el numero a la izquierda de `e` y lo multiplica por `10` elevado al poder del numero despues de la `e`. En otras palabras, la expresion `1e6` es quivalente a `1 x 10^6`. La `e` significa exponenciacion.

[11]:
```
# Python tambien utiliza notacion exponencial para mostrar numeros flotantes␣
↪bien grandes
2000000000000000000000000000000000000000.0
```

[11]: 2e+41

[12]:
```
1e-4  # tambien podemos utilizar numeros negativos
```

[12]: 0.0001

La expresión anterior es interpretada como `10` elevado al poder `-4`, lo que es `1/10000` o `0.0001`.

Los numeros de punto flotante tienen un tamaño limite. El maximo valor depende del sistema / computadora, pero algo como `2e400` esta fuera de la capacidad de la mayoría de las computadoras. `2x10^400` es mas que el numero total de atomos en el universo (se estima que existen entre `10^77` y `10^80` atomos en el universo).

[13]:
```
# cuando llegamos al numero maximo punto flotante, Python retorna `inf`
2e400
```

[13]: inf

1.1.3 Ejercicios

1. Escribe un programa que crea 2 variables, num1 y num2. Estas dos variables tendran un literal de numero entero 25,000,000 escrito de dos distintas formas.
2. Escribe un programa que asigna un literal de punto-flotante 175000.0 a la variable `num` utilizando notacion exponencial y luego imprimera `num`

1.2 Operadores Aritmeticos y Expresiones

En esta seccion aprenderemos a hacer aritmetica basica con numeros en Python: * Suma * Resta * Multiplicacion * Division

[14]:
```
# suma
1 + 1
```

[14]: 2

[15]:
```
# la suma de un numero punto-flotante con numero entero retorna punto-flotante
1.0 + 1
```

[15]: 2.0

[16]:
```
# resta
1 - 1
```

[16]: 0

[17]:
```
1.0 - 1
```

[17]: 0.0

[18]:
```
1.0 - 5
```

[18]: -4.0

[19]:
```
1 - -4
```

[19]: 5

[20]:
```
# multiplicacion
3 * 3
```

[20]: 9

[21]:
```
2 * 4.0
```

[21]: 8.0

[22]:
```
# division
9 / 3  # siempre retorna punto flotante
```

[22]: 3.0

[23]:
```
int(9 / 3)  # tenemos que convertirlo en numero entero
```

[23]: 3

[24]:
```
# sin embargo, int() elimina todos los decimales
int(5.0 / 2)
```

[24]: 2

[25]:
```
# podemos observar este comportamiento asi
int(2.5)
```

[25]: 2

[26]:
```
# division de numeros enteros
9 // 3
```

[26]: 3

[27]:
```
5.0 // 2
```

[27]:
```
2.0
```

[28]:
```
-3 // 2  # porque?
```

[28]:
```
-2
```

El operador // primero divide el numero a la izquierda por el numero de la derecha y luego redondea el resultado. Es por eso que -3 entre 2 es -1.5 y luego redondeamos -1.5 a -2.

[30]:
```
# dvision por 0
1 / 0
```

```
␣
↪---------------------------------------------------------------------------

       ZeroDivisionError                         Traceback (most recent call␣
↪last)

       <ipython-input-30-488f22d9f54d> in <module>
         1 # dvision por 0
    ----> 2 1 / 0


       ZeroDivisionError: division by zero
```

1.2.1 Exponenciacion

[31]:
```
2 ** 2
```

[31]:
```
4
```

[32]:
```
2 ** 3
```

[32]:
```
8
```

[33]:
```
2 ** 4
```

[33]:
```
16
```

[34]:
```
# pueden ser floats tambien
3 ** 1.5
```

[34]:
```
5.196152422706632
```

[35]:
```
9 ** 0.5
```

[35]: 3.0

[36]:
```
# exponenciacion negativa
2 ** -1
```

[36]: 0.5

La exponenciacion negativa es lo mismo que dividir `1` por el numero y su exponenciacion, es decir `2 ** -1` es lo mismo que `1 / (2 ** 1)`, lo cual es lo mismo a `1 / 2` o `0.5`. Similarmente, `2 ** -2` es lo mismo que `1 / (2 ** 2)`, lo cual es lo mismo a `1 / 4` o `0.25`.

1.2.2 El operador modulus %

Este operador `%`, llamado `modulus`, retorna lo que sobre de la division entre el numero de la izquierda del operador y el numero a la derecha

[37]:
```
5 % 3  # 5 // 3 es 1 y sobra 2
```

[37]: 2

[38]:
```
20 % 7  # 20 // 7 es 2 y sobra 6
```

[38]: 6

[40]:
```
16 % 8
```

[40]: 0

Uno de los casos de uso mas comunes de `%` es determinar si un numero es divisble por otro numero. Por ejemplo, un numero `n` es divisible unicacamente si `n % 2` resulta `0`.

1.2.3 Expresiones Aritmeticas

Podemos combinar operadores para formar expresiones complejas. Una **expresion** es una combinacion de numeros, operadores y parentesis sobre los cuales Python puede evaluar y retornar un valor.

[42]:
```
2 * 3 - 1
```

[42]: 5

[43]:
```
4/2 + 2**3
```

[43]: 10.0

[44]:
```
-1 + (-3 * 2 + 4)
```

[44]: -3

Las reglas sobre evaluacion de expresiones son las mismas que se enseñan en la escuela bajo el concepto de 'orden de operaciones'. Los operadores `*`, `/`, `//` y `%` tienen todos igual valor de precendia o prioridad en una expresion, y cada uno de estos tiene prioridad sobre los oepradores `+` o `-`. Esta es la razón por la cual `2 * 3 - 1` retorna `5` y no `4`, toda vez que la multiplicacion se calcula primero.

1.3 Reto: Realiza calculaciones con el dato de entrada del Usuario

Escribe un script llamado `exponente.py` que recibe 2 numeros del usuario y muestra el primer numero exponenciado al segundo numero. Por ejemplo:

```
Escribe una base: 1.2
Escribe un exponente: 3
1.2 exponenciado a 3 es 1.7279999999999998
```

En resumen: * Utilizaremos input() 2 veces, y su valor es asignado a dos variables * input() retorna un string, pero necesitamos que sea numeros * podemos utilizar **f-string** para imprimir el resultado * podemos asumir que el usuario insertara numeros como dato de entrada

1.4 Haz que Python mienta

Cual es el resultado de `0.1 + 0.2`? Es `0.3`?

[47]:
```
0.1 + 0.2
```

[47]:
```
0.30000000000000004
```

Esto no es un error de programa, sino un error conocido como error representacional de punto-flotante. El error esta relacionado a como los numeros son guardados en la memoria de una computadora. El numero `1.0` puede ser representado como una fracción `1/10`. Ambos numeros `0.1` y `1/10` son **representaciones decimales** / **representaciones de base 10**. Las computadoras, sin embargo, guardan los puntos flotantes en representacion de base 2, mas comunmente conocido como **representacion binaria**.

Cuando la representacion es binaria, algo familiar e inesperado sucede con el numero decimal `0.1`. La fraccion `1/3` no tiene una representacion decimal. Es decir, `1/3 = 0.3333...` con una infidad de `3`s despues del punto decimal. Lo mismo sucede con la fraccion `1/10` en binario.

La representacion binaria de `1/10` es la siguiente fraccion, repetida indefinidamente: `0.0001100110011001100110011001100110011...`

Las computadoras tienen memoria finita, no infinita, por tanto el numero `0.1` debe ser guardado como una aproximacion y no con su valor correcto. La aproximacion que se guarda es un poco mas alta que el valor actual y se ve algo asi: `0.1000000000000000055511151231257827021181583404541015625`.

[49]:
```
# sin embargo, cuando mostramos 0.1 en Python
0.1
```

[49]:
```
0.1
```

Lo que sucede es que la aproximacion de `0.1` en binario es solo una aproximacion, y es posible que mas de un numero decimal tenga la misma aproximacion binaria. Por ejemplo, los numeros `0.1`

y `0.10000000000000001` tienen la misma aproximacion binaria. Python imprime el decimal mas corto que comparte esa aproximacion.

Esto explica la razon por la cual `0.1 + 0.2` no es igual a `0.3`, toda vez que Python agrega la aproximacion binaria de `0.1` y `0.2`, lo cual resulta en un numero que **no es** la aproximacion binaria de `0.3`.

Esta informacion es de fundamental importancia si estamos programando aplicaciones para finanzas o computacion cientifica.

1.5 Funciones matematicas y metodos de numeros

Python tiene algunas funciones incorporadas para trabajar con numeros. En esta seccion aprenderemos sobre los tres metodos mas comunes para trabajar con numeros: 1. `round()`, para redondear numeros 2. `abs()`, para obtener el valor absoluto de un numero 3. `pow()`, para exponenciar un numero

1.5.1 round()

[50]:
```
round(2.3)
```

[50]: 2

[51]:
```
round(2.7)
```

[51]: 3

[52]:
```
# comportamiento inesperado
round(2.5)
```

[52]: 2

[53]:
```
# comportamiento inesperado
round(3.5)
```

[53]: 4

Python redondea numeros conforme a una estrategia llamada **rounding ties to even** o **redondear empates al par**. Un **empate** es un numero cuyo ultimo digito es `5`. Por tanto, `2.5` y `3.1415` son empates, pero no `1.37`.

Cuando reondeamos empates al par, primero observamos el digito ubicado un decimal a la izquiera del ultimo digito en el empate. Si ese digito es par, redondeamos hacia abajo, y si es impar, redondeamos hacia arriba. Es por eso que `2.5` redondea hacia abajo y `3.5` redondea hacia arriba.

Esta estrategia de redondear es recomendada para numeros de punto flotante por el Instituto de Ingenieros Electricos y Electronicos (**IEEE**, por sus siglas en ingles) porque ayuda a limitar el impacto de redondear en operaciones que contienen muchos numeros. Fue publicado en 1985.

[1]:
```
# podemos redondear a un cierto numero de decimales
round(3.14159, 3)
```

[1]:
```
3.142
```

[2]:
```
# pero el segundo argumento debe ser un numero entero
round(2.65, 1.4)
```

```
␣
↪---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call␣
↪last)

<ipython-input-2-98a21b0df6f1> in <module>
  1 # pero el segundo argumento debe ser un numero entero
----> 2 round(2.65, 1.4)


TypeError: 'float' object cannot be interpreted as an integer
```

[4]:
```
# valores esperado: 2.68
round(2.675, 2)
```

[4]:
```
2.67
```

La estrategia de redondear empates al par exige un resultado de `2.68` en la operacion anterior, sin embargo el resultado fue `2.67`. Este error es el resultado de lo que mencionamos anteriormente cuando hablamos del error representacional de los numeros de punto flotante.

Lidear con numeros de punto flotante puede ser frustrante, pero la frustracion no es especifica a Python. Todos los lenguajes que implementan el estandar de numeros punto flotante `float` de la IEEE tienen este problema, incluyendo C/C++, Java y JavaScript.

En la mayoria de los casos, estos pequeños erores no causan gran impacto y el resultado de `round()` es util.

1.5.2 abs()

[57]:
```
abs(3)
```

[57]:
```
3
```

[58]:
```
# abs siempre retorna un numero positivo
abs(-5.0)
```

[58]:
```
5.0
```

1.5.3 pow()

[59]:
```
# ademas de utilizar la nomenclatura 3 ** 3, podemos utilizar pow()
pow(3, 3)
```

[59]:
```
27
```

[60]:
```
# tambien puede ser con exponenciacion negativa
pow(2, -2)
```

[60]:
```
0.25
```

Cual es la diferencia entre `**` y `pow()`? Esta ultima acepta un tercer argumento / parametro que computa el primer numero exponenciado al poder del segundo numero y luego toma el modulo `%` con respecto al tercer numero.

En otras palabras, `pow(x, y, z)` es equivalente a `(x ** y) % z`

[61]:
```
pow(2, 3, 2)
```

[61]:
```
0
```

[63]:
```
8 % 2
```

[63]:
```
0
```

1.5.4 Revisa si un numero flotante es entero

Los metodos de numeros no son utilizados frecuentemes, pero hay uno que puede ser util. Los numeros de tipo flotante tienen un metodo `.is_integer()` que retorna `True` si el numero es **entero** o `False` de lo contrario. El numero es entero cuando no tiene ninguna parte fraccionada.

[5]:
```
num = 2.5
num.is_integer()
```

[5]:
```
False
```

[6]:
```
num = 2.0
num.is_integer()
```

[6]:
```
True
```

1.5.5 Ejercicios

1. Escribe un script que solicita al usuario un numero y luego muestra ese numero redondeado a dos puntos decimales.

2. Escribe un script que solicita al usuario un numero y luego muestra el valor absoluto de ese numero.
3. Escribe un script que solicita al usuario dos numeros utilizando input dos veces y luego muestra si la diferencia entre esos dos numeros es un numero entero.

1.6 Imprime numeros con estilo

Mostrar numeros al usuario requiere insertar numeros a un string. En la seccion de strings, aprendimos hacer esto con **f-strings**.

[66]:
```
n = 7.125
f'El valor de n es {n}'
```

[66]:
```
'El valor de n es 7.125'
```

Las llaves `{}` soportan un lenguaje de formato que podemos utilizar para alterar la apariencia del valor final del string. Por ejemplo, para cambiar el formato del valor de `n` a 2 valores decimales, reemplazamos los contenidos de las llaves en el f-string con `{n:.2f}`

[67]:
```
f'El valor de n es {n:.2f}'
```

[67]:
```
'El valor de n es 7.12'
```

El colon `:` despues de la variable `n` indica que todo despues del mismo es parte de una especificacion de formato. En este ejemplo, la especificacion de formato es `.2f`. El `.2` en `.2f` redondea el numero a dos puntos decimales y la `f` le dice a Python que muestre `n` como un numero de punto flotante. Esto significa que el numero se muestra con exactamente 2 puntos decimales aunque el numero original tiene menos puntos decimales.

[68]:
```
# Python de todas formas utiliza la estrategia de redondear empates al par en␣
↪este caso tambien
n = 7.126
f'El valor de n es {n:.2f}'
```

[68]:
```
'El valor de n es 7.13'
```

[69]:
```
# un punto decimal
f'El valor de n es {n:.1f}'
```

[69]:
```
'El valor de n es 7.1'
```

[70]:
```
n = 1
f'El valor de n es {n:.2f}'
```

[70]:
```
'El valor de n es 1.00'
```

[71]:
```
f'El valor de n es {n:.5f}'
```

[71]:
```
'El valor de n es 1.00000'
```

[72]:
```
# podemos usar comas para agrupar partes de un numero entero grande
n = 1234567890
f'El valor de n es {n:,}'
```

[72]:
```
'El valor de n es 1,234,567,890'
```

[73]:
```
# agrupemos un numero entero con decimal
n = 1234.56
f'El valor de n es {n:,.2f}'
```

[73]:
```
'El valor de n es 1,234.56'
```

[74]:
```
# la especificacion `,.2f` es util para mostrar dinero
balance = 2000.0
gastado = 256.35
restante = balance - gastado
f'Despues de gastar ${gastado:.2f}, me quedé con ${restante:,.2f}'
```

[74]:
```
'Despues de gastar $256.35, me quedé con $1,743.65'
```

Otra opción util es el porcentaje `%`, utilizado para mostrar porcentajes. Esta opcion multiplica el numero por `100` y lo muestra en formato de punto flotante, seguido por el simbolo porcentual. Siempre debe estar al final de la especificacion y no se puede mezclar con otra especificacion de formato.

[7]:
```
ratio = 0.9
f'Mas del {ratio:.1%}'
```

[7]:
```
'Mas del 90.0%'
```

Las especificaciones de formato son poderosas y extensas. Aqui solo explicamos lo basico. Le invito a leerse la documentacion oficial al respecto: https://docs.python.org/3/library/string.html#format-string-syntax

1.6.1 Ejercicios

1. Imprime el resultado de la calculacion `3 ** .125` como numero de punto flotante `float` con tres puntos decimales.
2. Imprime el numero `150000` como moneda / dinero y con los miles agrupados con commas, a dos puntos deicmales.
3. Imprime el resultado de `2 / 10` como porcentaje sin ningun punto decimal. El dato de salida debe ser como `20%`.

1.7 Resumen

En esta seccion aprendimos a trabajar con numeros en Python. Vimos que existen dos tipos basicos de numeros: numeros enteros y numeros de punto flotante.

Aprendimos a hacer aritmetica basica con numeros utilizando los operadores `+`, `-`, `*`, `/` y `%`. Tambien vimos como escribir expresiones aritmeticas y aprendimos algunas de las mejores practicas para formatear expresiones aritmeticas.

Despues aprendimos sobre los numeros de punto flotante y como no son 100% precisos. Esta limitacion no es relativa a Python, sino al hecho de como la industria ha resuelto el problema de guardar estos numeros en la memoria de la computadora.

Ademas, vimos como redondear numeros a ciertos puntos decimales con `round()` y aprendimos que este metodo utiliza la estrategia de redondear empates al par, lo cual es diferente a como aprendimos a redondear numeros en la escuela.

Finalmente, vimos numerosas maneras en como cambiar el formato especifico de un numero.