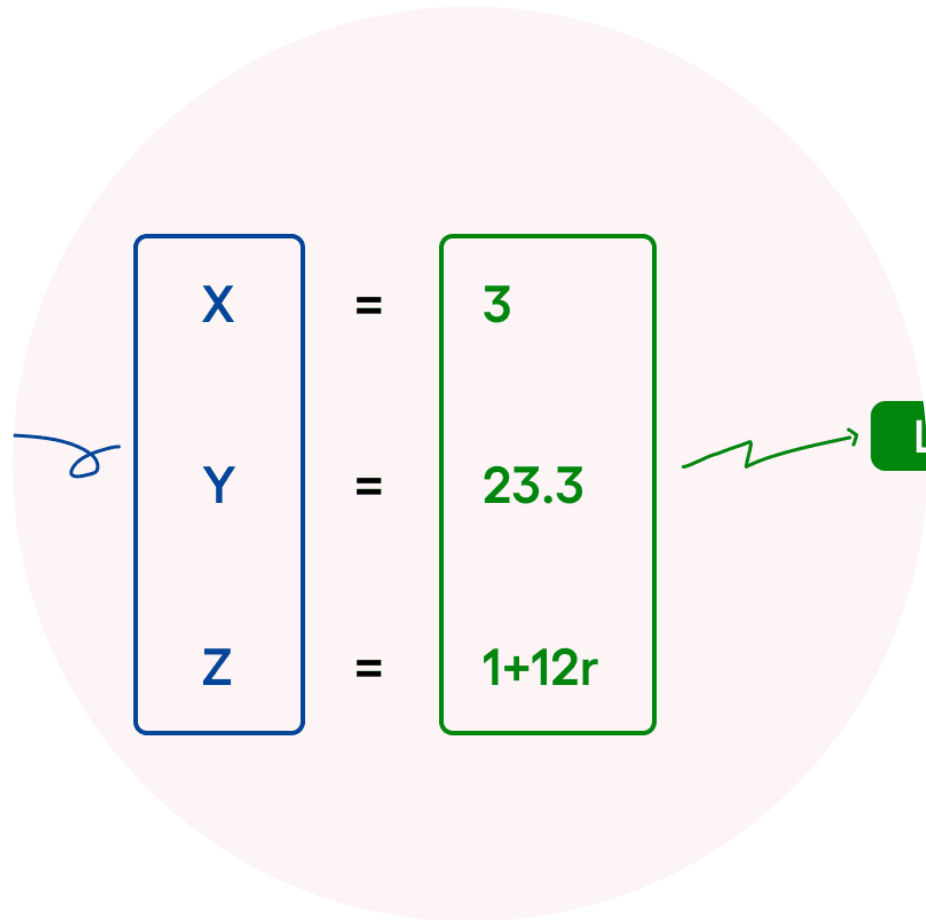


LITERALES

Los datos en si mismos





UN LITERAL SE REFIERE A DATOS CUYOS VALORES ESTÁN DETERMINADOS POR EL LITERAL MISMO.

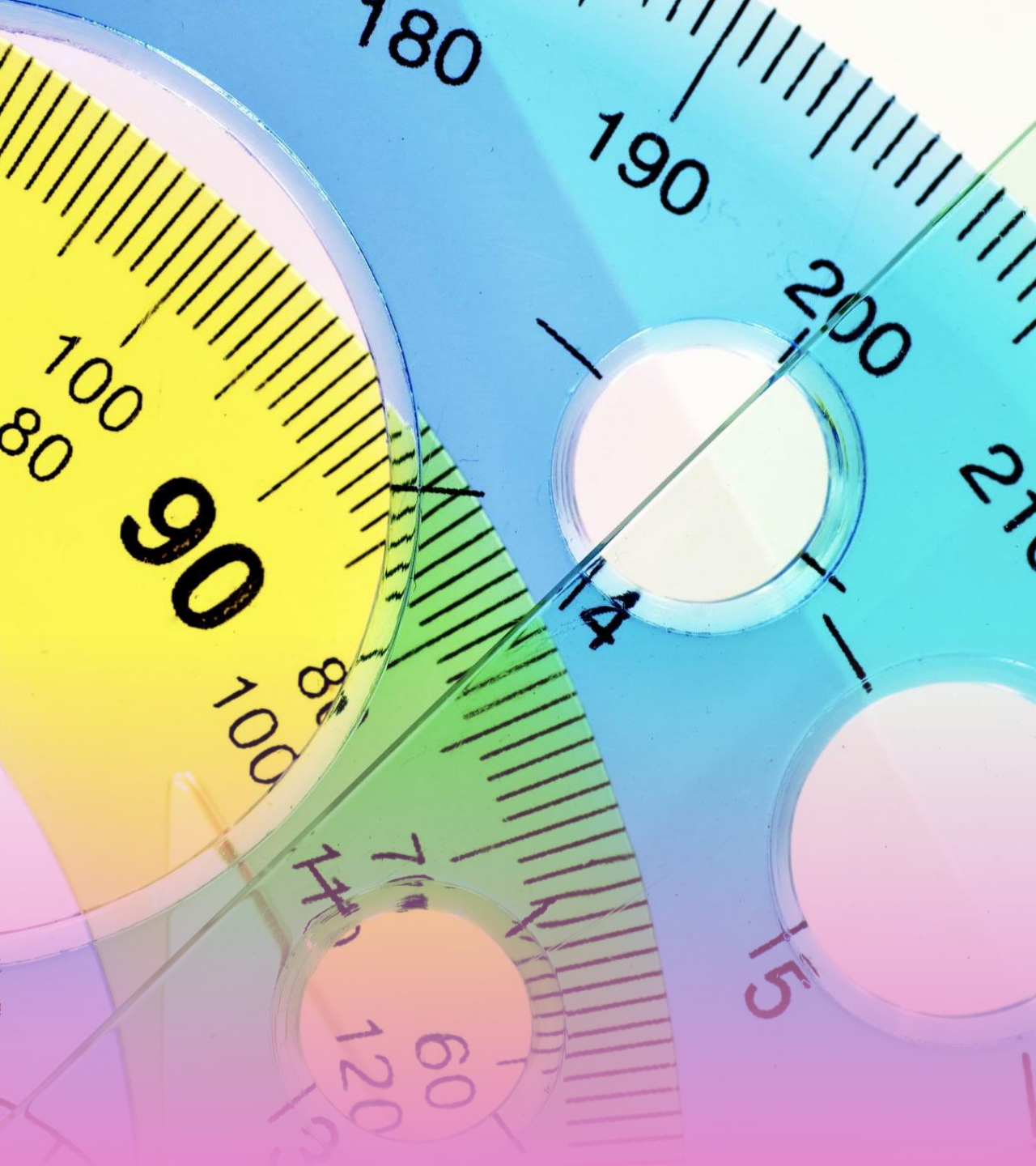
- Ahora que tienes un poco de conocimiento acerca de algunas de las poderosas características que ofrece la función `print()`, es tiempo de aprender sobre cuestiones nuevas, y un nuevo término - el literal.



VARIABLE

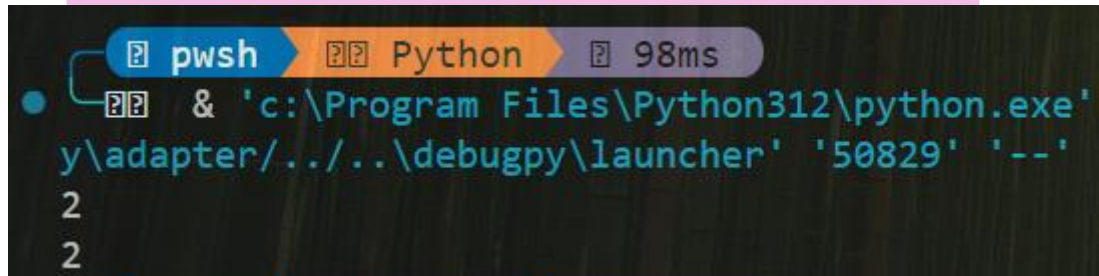
DEBIDO A QUE ES
UN CONCEPTO UN
POCO DIFÍCIL DE
ENTENDER, UN
BUEN EJEMPLO
PUEDE SER MUY
ÚTIL.

- Observa los siguientes dígitos:
- 123
- ¿Puedes adivinar qué valor representa?
Claro que puedes - es ciento veintitrés.
- Que tal este:



- `c`
- ¿Representa algún valor? Tal vez. Puede ser el símbolo de la velocidad de la luz, por ejemplo. También puede representar la constante de integración. Incluso la longitud de una hipotenusa en el Teorema de Pitágoras. Existen muchas posibilidades.
- No se puede elegir el valor correcto sin algo de conocimiento adicional.
- Y esta es la pista: `123` es un literal, y `c` no lo es.
- Se utilizan literales para codificar datos y ponerlos dentro del código. Ahora mostraremos algunas convenciones que se deben seguir al utilizar Python.

```
print("2")  
print(2)
```



```
pwsh Python 98ms  
& 'c:\Program Files\Python312\python.exe'  
y\adapter/../../debugpy\launcher' '50829' '--'  
2  
2
```

Comencemos con un sencillo experimento, observa el fragmento de código.

La primera línea luce familiar. La segunda parece ser errónea debido a la falta visible de comillas.

Intenta ejecutarlo.

Si todo salió bien, ahora deberías de ver dos líneas idénticas.

¿Qué paso? ¿Qué significa?

A través de este ejemplo, encuentras dos tipos diferentes de literales:

Una cadena, la cual ya conoces.

Y un número entero, algo completamente nuevo.

La función `print()` los muestra exactamente de la misma manera. Sin embargo, internamente, la memoria de la computadora los almacena de dos maneras completamente diferentes. La cadena existe como eso, solo una cadena, una serie de letras.

El número es convertido a una representación máquina (una serie de bits). La función `print()` es capaz de mostrar ambos en una forma legible para humanos.

Vamos a tomar algo de tiempo para discutir literales numéricas y su vida interna.

ENTEROS

Quizá ya sepas un poco acerca de cómo las computadoras hacen cálculos con números. Tal vez has escuchado del sistema binario, y como es que ese es el sistema que las computadoras utilizan para almacenar números y como es que pueden realizar cualquier tipo de operaciones con ellos.



Enteros vs. Flotantes

```
>>> entero = 2      # este es un int
>>> flotante = 2.0  # este es un float
>>> entero
2
>>> flotante
2.0
>>> type(entero)
<class 'int'>
>>> type(flotante)
<class 'float'>
>>> |
```

```
>>> type(1e5)
<class 'float'>
>>> type(100000)
<class 'int'>
```

No exploraremos las complejidades de los sistemas numéricos posicionales, pero se puede afirmar que todos los números manejados por las computadoras modernas son de dos tipos:

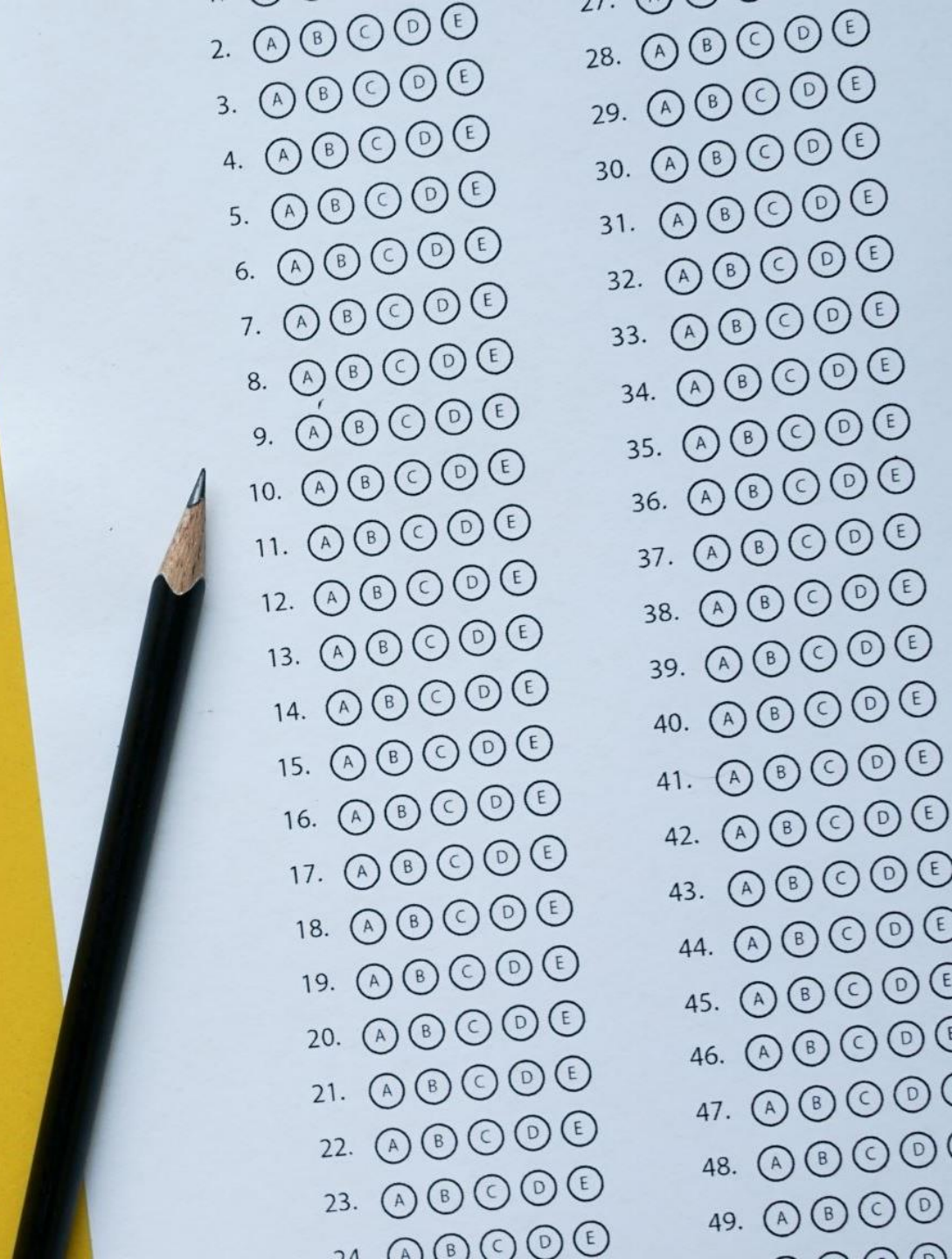
- **Enteros**, es decir, aquellos que no tienen una parte fraccionaria.
- Y números **punto-flotantes** (o simplemente **flotantes**), los cuales contienen (o son capaces de contener) una parte fraccionaria.

Esta definición no es tan precisa, pero es suficiente por ahora. La distinción es muy importante, y la frontera entre estos dos tipos de números es muy estricta. Ambos tipos difieren significativamente en como son almacenados en una computadora y en el rango de valores que aceptan.



- La característica del valor numérico que determina el tipo, rango y aplicación se denomina el **tipo**.
- Si se codifica un literal y se coloca dentro del código de Python, la forma del literal determina la representación (tipo) que Python utilizará para **almacenarlo en la memoria**.
- Por ahora, dejemos los números flotantes a un lado (regresaremos a ellos pronto) y analicemos como es que Python reconoce un número entero.

- El proceso es casi como usar lápiz y papel, es simplemente una cadena de dígitos que conforman el número, pero hay una condición, no se deben insertar caracteres que no sean dígitos dentro del número.
- Tomemos por ejemplo, el número once millones ciento once mil ciento once. Si tomaras ahorita un lápiz en tu mano, escribirías el siguiente número: 11,111,111, o así: 11.111.111, incluso de esta manera: 11 111 111.



¿QUÉ SIGNIFICA?

Lo que está describiendo es un proceso de escribir un número grande usando separadores visuales, como comas, puntos o espacios, para facilitar su lectura. Este método es común para hacer que los números largos sean más manejables y fáciles de interpretar a simple vista.

Esto significa que, aunque en la práctica puedes usar estos separadores para visualizar el número más claramente, cuando lo codifiques o lo representes en un sistema que requiera únicamente dígitos, deberás eliminar esos separadores y asegurarte de que solo se usen números (0-9) en la cadena.





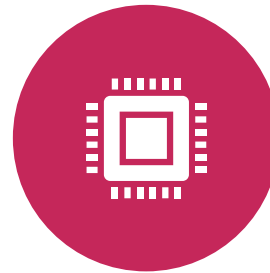
Es claro que la separación hace que sea más fácil de leer, especialmente cuando el número tiene demasiados dígitos. Sin embargo, Python no acepta estas cosas. Está prohibido. ¿Qué es lo que Python permite? El uso de guion bajo en los literales numéricos.*



Por lo tanto, el número se puede escribir ya sea así: 11111111, o como sigue: 11_111_111.

NOTA *Python 3.6 ha introducido el guion bajo en los literales numéricos, permitiendo colocar un guion bajo entre dígitos y después de especificadores de base para mejorar la legibilidad. Esta característica no está disponible en versiones anteriores de Python.

NEGATIVOS?



¿CÓMO SE CODIFICAN LOS
NÚMEROS NEGATIVOS EN
PYTHON? COMO NORMALMENTE
SE HACE, AGREGANDO UN
SIGNO DE MENOS. SE PUEDE
ESCRIBIR: -11111111, O -
11_111_111.



LOS NÚMEROS POSITIVOS NO
REQUIEREN UN SIGNO POSITIVO
ANTEPUESTO, PERO ES
PERMITIDO, SI SE DESEA HACER.
LAS SIGUIENTES LÍNEAS
DESCRIBEN EL MISMO NÚMERO:
+11111111 Y 11111111



```
1 b = 0b01010 # BINARIO
2 o = 0o1232 # OCTAL
3 d = 1234124 # DECIMAL
4 h = 0x1f2a # HEXADECIMAL
5
6 sum = b + o + d + h # INT
```

ENTEROS: NÚMEROS OCTALES Y HEXADECIMALES

Octal Number System

4	2	1	Octal Representation
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

100	111	010	011	010
↓	↓	↓	↓	↓
4	7	2	3	2

(767) ₈	?
?	(440) ₈

51


COA

Existen dos convenciones adicionales en Python que no son conocidas en el mundo de las matemáticas. El primero nos permite utilizar un número en su representación **octal**.

Si un número entero esta precedido por un código 00 o 0o (cero-o), el número será tratado como un valor octal. Esto significa que el número debe contener dígitos en el rango del [0..7] únicamente.

0o123 es un número **octal** con un valor (decimal) igual a 83.

La función print() realiza la conversión automáticamente. Intenta esto:

```
literales >  octal.py  
1 print(0o123)
```

```
ms-python.debugpy-2024.10.0-win32  
ecap\Python\literales\octal.py'  
83
```

```
1 print(0x123)
```

```
ecap\Python\literales\octal.py'  
291
```

La segunda convención nos permite utilizar números en **hexadecimal**. Dichos números deben ser precedidos por el prefijo 0x o 0X (cero-x).

0x123 es un número **hexadecimal** con un valor (decimal) igual a 291. La función print() puede manejar estos valores también.

FLOTANTES

- Ahora es tiempo de hablar acerca de otro tipo, el cual esta designado para representar y almacenar los números que (como lo diría un matemático) tienen una **parte decimal no vacía**.
- Son números que tienen (o pueden tener) una parte fraccionaria después del punto decimal, y aunque esta definición es muy pobre, es suficiente para lo que se desea discutir.
- Cuando se usan términos como *dos y medio* o *menos cero punto cuatro*, pensamos en números que la computadora considera como números **punto-flotante**:

2.5

-0.4

NOTA

- Nota: *dos punto cinco* se ve normal cuando se escribe en un programa, sin embargo si tu idioma nativo prefiere el uso de una coma en lugar de un punto, se debe asegurar que **el número no contenga comas**.
- Python no lo aceptará, o (en casos poco probables) puede malinterpretar el número, debido a que la coma tiene su propio significado en Python.
- Si se quiere utilizar solo el valor de dos punto cinco, se debe escribir como se mostró anteriormente. Nota que hay un punto entre el 2 y el 5, no una coma.

Como puedes imaginar, el valor de cero punto cuatro puede ser escrito en Python como:

0.4

Pero no hay que olvidar esta sencilla regla, se puede omitir el cero cuando es el único dígito antes del punto decimal.

En esencia, el valor 0.4 se puede escribir como:

.4

Por ejemplo: el valor de 4.0 puede ser escrito como:

4.

Esto no cambiará su tipo ni su valor.

0.0

El punto decimal es esencialmente importante para reconocer números punto-flotantes en Python.

Observa estos dos números:

4 4.0



**ENTEROS FRENTE
A FLOTANTES**



Se puede pensar que son idénticos, pero Python los ve de una manera completamente distinta.



4 es un número entero, mientras que 4.0 es un número punto-flotante.



El punto decimal es lo que determina si es flotante.



Por otro lado, no solo el punto hace que un número sea flotante. Se puede utilizar la letra e.



Cuando se desea utilizar números que son muy pequeños o muy grandes, se puede implementar la notación científica.

Por ejemplo, la velocidad de la luz, expresada en metros por segundo. Escrita directamente se vería de la siguiente manera: 300000000.

Para evitar escribir tantos ceros, los libros de texto emplean la forma abreviada, la cual probablemente hayas visto: 3×10^8 .

Se lee de la siguiente manera: tres por diez elevado a la octava potencia.

En Python, el mismo efecto puede ser logrado de una manera similar, observa lo siguiente:

3E8

La letra E (también se puede utilizar la letra minúscula e - proviene de la palabra exponente) la cual significa por diez a la n potencia.

Nota:

El exponente (el valor después de la E) debe ser un valor entero.

La base (el valor antes de la E) puede o no ser un valor entero.

Codificando Flotantes

Veamos ahora como almacenar números que son muy pequeños (en el sentido de que están muy cerca del cero).

Una constante de física denominada "*La Constante de Planck*" (denotada como h), de acuerdo con los libros de texto, tiene un valor de: 6.62607×10^{-34} .

Si se quisiera utilizar en un programa, se debería escribir de la siguiente manera:

```
6.62607E-34
```

Nota: el hecho de que se haya escogido una de las posibles formas de codificación de un valor flotante no significa que Python lo presentará de la misma manera.

Python podría en ocasiones elegir una notación diferente.

EJEMPLO

Por ejemplo, supongamos que se ha elegido utilizar la siguiente notación:

```
0.000000000000000000000001
```

Cuando se corre en Python:

```
print(0.000000000000000000000001)
```

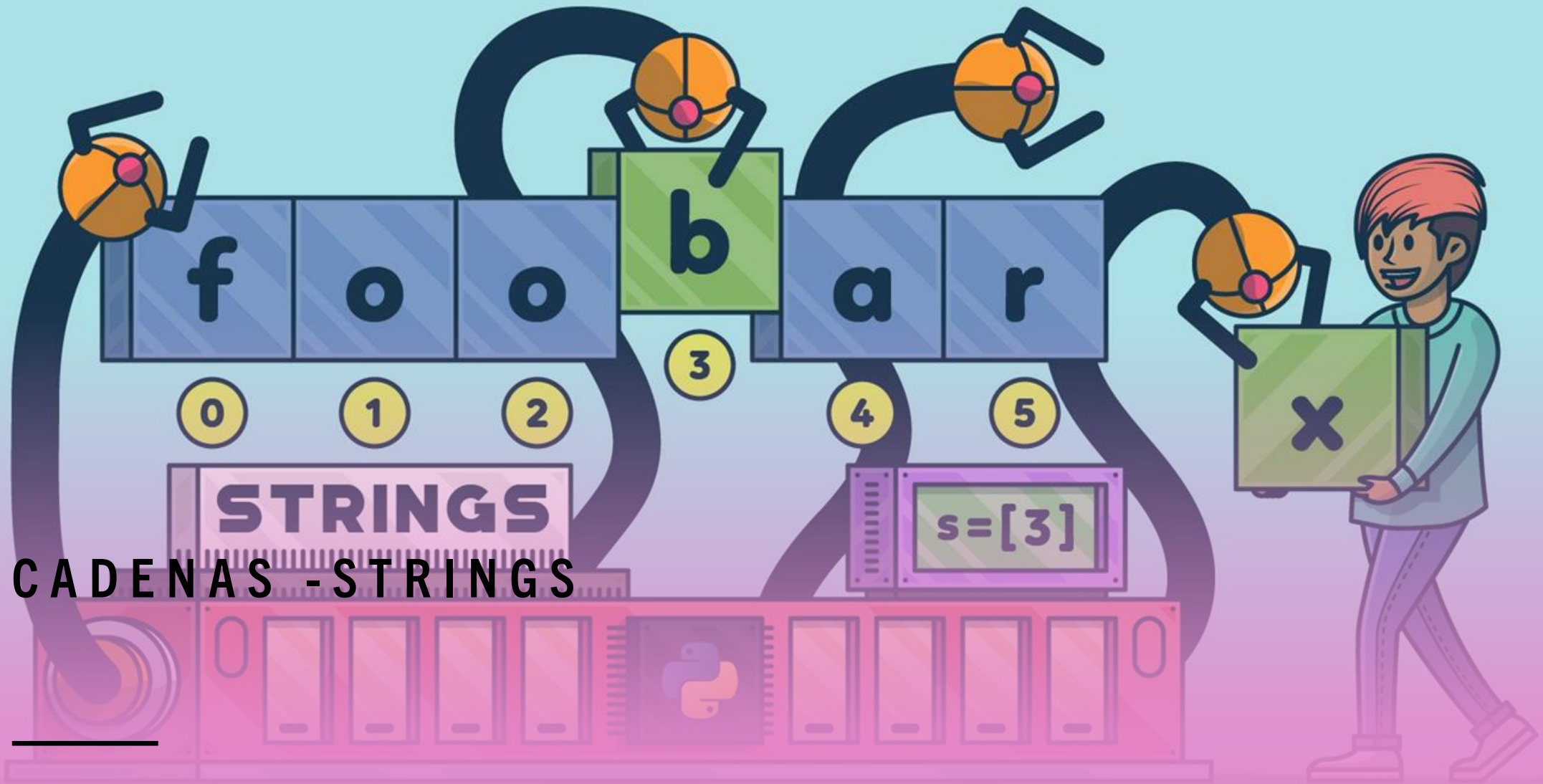
Este es el resultado:

```
1e-22
```

salida

Python siempre elige **la presentación más corta del número**, y esto se debe de tomar en consideración al crear literales.



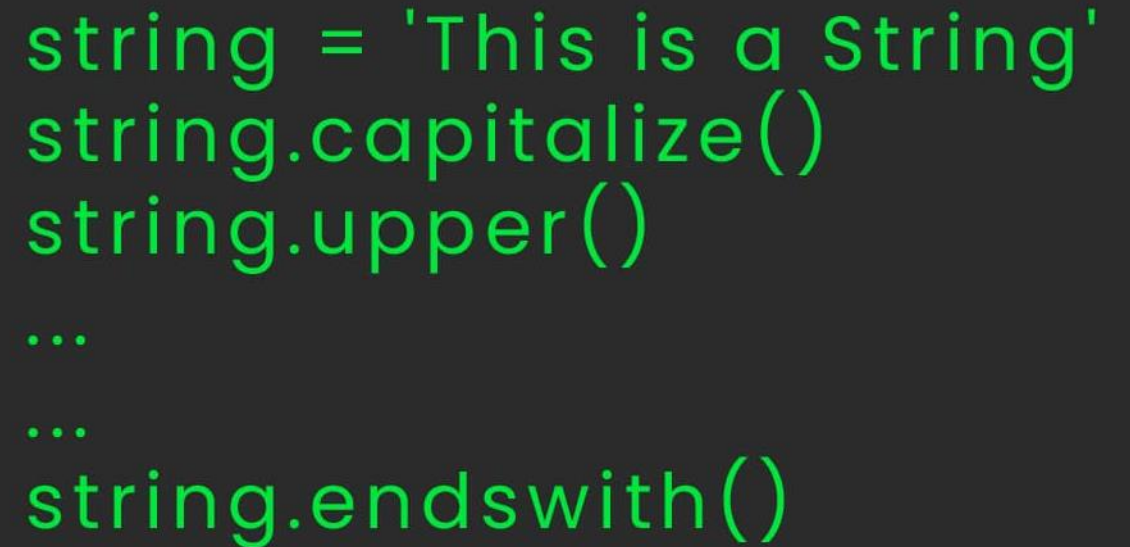


CADENAS - STRINGS

Las cadenas se emplean cuando se requiere procesar texto (como nombres de cualquier tipo, direcciones, novelas, etc.), no números.

Ya conoces un poco acerca de ellos, por ejemplo, que las cadenas requieren comillas así como los flotantes necesitan punto decimal.

Este es un ejemplo de una cadena: "Yo soy una cadena."



```
string = 'This is a String'  
string.capitalize()  
string.upper()  
...  
...  
string.endswith()
```



Sin embargo, hay una cuestión. ¿Cómo se puede codificar una comilla dentro de una cadena que ya está delimitada por comillas?



Supongamos que se desea mostrar un muy sencillo mensaje:



Me gusta "Monty Python"




¿Cómo se puede hacer esto sin generar un error? Existen dos posibles soluciones.

La primera se basa en el concepto ya conocido del **carácter de escape**, el cual recordarás se utiliza empleando la **diagonal invertida**. La diagonal invertida puede también escapar de la comilla. Una comilla precedida por una diagonal invertida cambia su significado, no es un limitador, simplemente es una comilla. Lo siguiente funcionará como se desea:

```
print("Me gusta \"Monty Python\"")
```

Nota: Las combinaciones de caracteres que constan de una barra diagonal inversa (\) seguidas de una letra o de una combinación de dígitos se denominan "secuencias de escape". Para representar un carácter de nueva línea, comillas simples u otros caracteres en una constante de caracteres, debe usar secuencias de escape.

La segunda solución puede ser un poco sorprendente. Python puede utilizar **una apóstrofe en lugar de una comilla**. Cualquiera de estos dos caracteres puede delimitar una cadena, pero para ello se debe ser **consistente**.



Si se delimita una cadena con una comilla, se debe cerrar con una comilla.



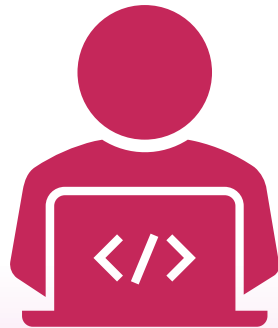
Si se inicia una cadena con un apóstrofe, se debe terminar con un apóstrofe.



Este ejemplo funcionará también:

```
print('Me gusta "Monty Python"')
```


VALORES BOOLEANOS



Para concluir con los literales de Python, existen dos más.



No son tan obvios como los anteriores y se emplean para representar un valor muy abstracto - **la veracidad**.

Cada vez que se le pregunta a Python si un número es más grande que otro, el resultado es la creación de un tipo de dato muy específico - un valor booleano.

El nombre proviene de George Boole (1815-1864), el autor de Las Leyes del Pensamiento, las cuales definen el Álgebra Booleana - una parte del álgebra que hace uso de dos valores: Verdadero y Falso, denotados como 1 y 0.

Un programador escribe un programa, y el programa hace preguntas. Python ejecuta el programa, y provee las respuestas. El programa debe ser capaz de reaccionar acorde a las respuestas recibidas.

Afortunadamente, las computadoras solo conocen dos tipos de respuestas:

- Si, esto es verdad.
- No, esto es falso.

```
>>> True or True
True
>>> False or True
True
>>> True or False
True
>>> False or False
False
>>>
```



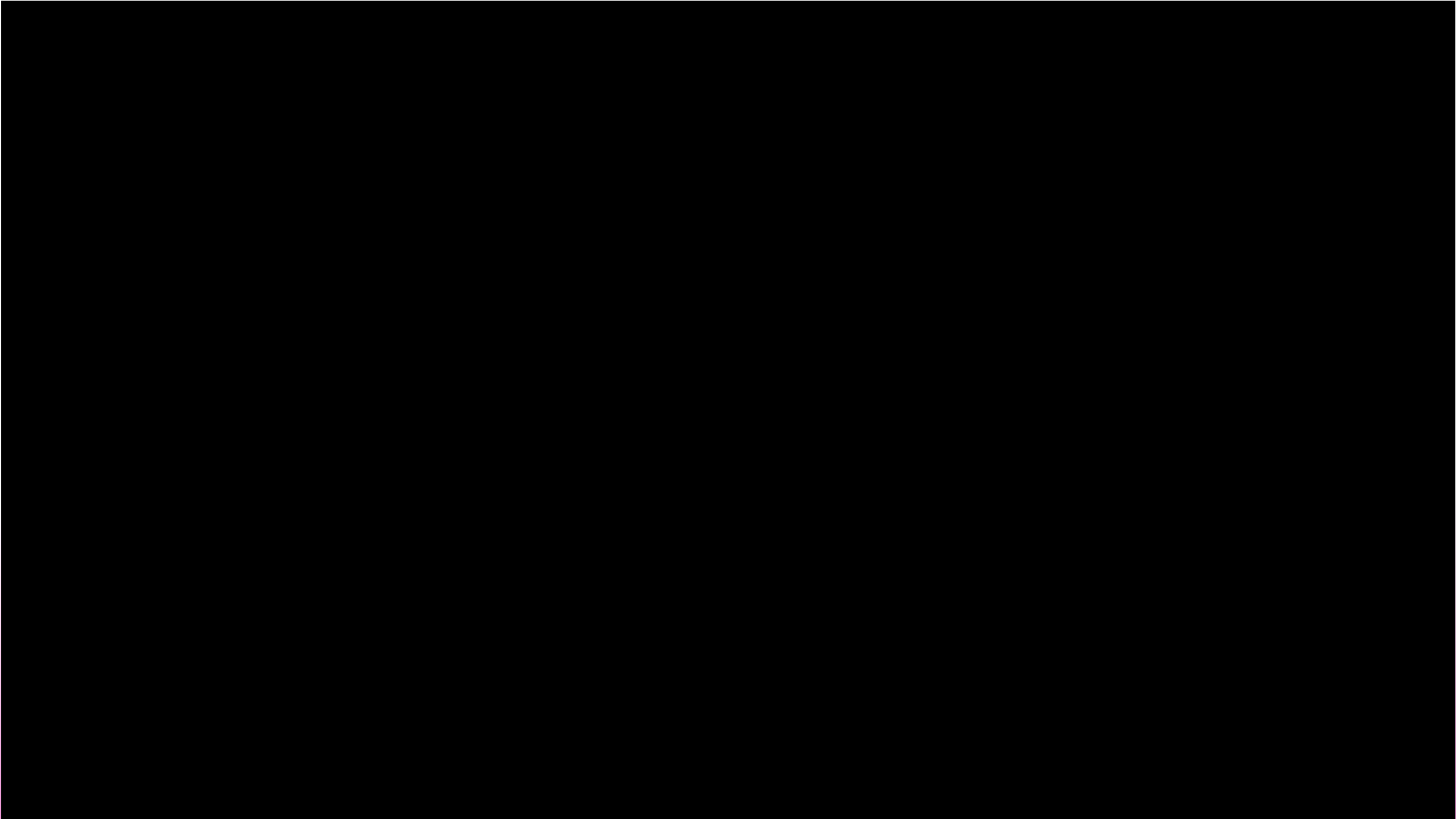
Python, es entonces, un reptil binario.

Estos dos valores booleanos tienen denotaciones estrictas en Python:

True

False

No se pueden cambiar, se deben tomar estos símbolos como son, incluso respetando las mayúsculas y minúsculas.



**GRACIAS POR SU
ATENCIÓN**



EJERCICIO: