Taller de Física Computacional

Literales y Operadores aritméticos

Cristián G. Sánchez y Carlos J. Ruestes

2021

Aclaración terminológica

Aclaración terminológica

La documentación oficial de Python está en inglés. Vamos a tratar de traducir las palabras importantes dentro de lo posible. A veces utilizaremos directamente en inglés para simplificarnos la existencia.

Tokens

Tokens

(Simplificando) un **programa** escrito en Python es leído por el intérprete como una secuencia de *tokens*.

Lista de *tokens*:

NUEVA LÍNEA, INDENTACIÓN, DEDENTACIÓN, identificadores, palabras clave, literales, operadores y delimitadores

Palabras clave

Palabras clave

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

Operadores

Operadores

+	-	*	**	/	//	%	@
<<	>>	&	|	^	~	:=	
<	>	<=	>=	==	!=		

Delimitadores

Delimitadores

(	)	[	]	{	}	
,	:	.	;	@	=	->
+=	-=	*=	/=	//=	%=	@=
&=	|=	^=	>>=	<<=	**=	

Identificadores o nombres

Identificadores o nombres

para más adelante

Literales

Literales

Los *literales* son una notación para valores **constantes** de **algunos** tipos incorporados en el lenguaje.

Tokens

Tokens

(Simplificando) un **programa** escrito en Python es leído por el intérprete como una secuencia de *tokens*.

Lista de *tokens*:

NUEVA LÍNEA, INDENTACIÓN, DEDENTACIÓN, identificadores, palabras clave, literales, operadores y delimitadores

Literales Numéricos

Enteros

- Decimales: `123431235` o `123_431_235`
- Binarios: `0b0101010101` o `0B0_1010_101_01` (puede ser
- Octales: `0o01234567` o `0O01_234_567`
- Hexadecimales: `0xFFFFFFFF` o `0XFF_FF_FF_FF`

Notas: Los caracteres `b`,`o` y `f` pueden ser mayúscula o minúscula. El guión bajo puede usarse para agrupar dígitos de forma arbitraria. Los enteros en python (versión 3 en adelante) no son acotados.

Literales Numéricos

Punto Flotante

- `3.14` o `10.` o `10.0` o `31.4e-1` o `31.4e5`

Nota: Los las cotas para los números de punto flotante dependen de la implementación.

Imaginarios

- `3.14j` o `10.j` o `10.0j` o `31.4e-1j` o `31.4e5j`

Nota: Los números complejos se construyen sumando un imaginario a un punto flotante real.

Literales de caracteres

Cadenas de caracteres

- `"Esto es una cadena"` o `'Esto es una cadena'`
- Se pueden construir cadenas multilínea con triples comillas:

```
"""Esto es
una cadena"""
```

o

```
'''Esto también es
una cadena'''
```

- Las *secuencias de escape* `\n \t \\ \' \"` se utilizan para insertar una nueva línea, un tabulador, una barra y los dos tipos de comillas respectivamente, dentro de una cadena de caracteres.

Literales de caracteres

Cadenas de formato

- Las cadenas de formato se utilizan para construir cadenas que contienen valores de expresiones que son evaluadas y a las que se les da formato en el momento de construcción de la cadena.
- Las cadenas de formato están precedidas por una `f` seguida de alguna comilla y contienen las expresiones a ser evaluadas y una especificación de formato entre llaves. Siguen algunos ejemplos:
- `f" uno y un tercio con tres decimales es {1 + 1/3:.3f}"`
- `f" mil en un espacio de 10 caracteres es {10*10*10:10d}"`
- La expresión de formato tiene muchas posibilidades, la documentación puede encontrarse en el manual.

Operadores aritméticos

Para poder empezar a hacer "algo" usando Python introducimos ahora los operadores aritméticos:

Operadores aritméticos

- `+`: Suma como binario o elemento positivo como unario.
- `-`: Resta como binario o negativo como unario.
- `*`: Multiplicación, es binario.
- `/`: División, es binario.
- `//`: División entera, es binario.
- `%`: Módulo (resto de la división entera), es binario.
- `**`: Exponenciación, es binario
- Se pueden utilizar paréntesis dentro de una expresión para asegurar el orden de evaluación.

Estos operadores son un subconjunto de los operadores que listamos anteriormente al hablar de *tokens*.

Detalles importantes

Promoción

- Una expresión compuesta sólo por enteros (incluyendo cualquier resultado intermedio) da como resultado un entero.
- La presencia de un número de punto flotante convierte a toda la expresión a punto flotante.
- La presencia de un número complejo convierte a toda la expresión en compleja.

Consejo

Nunca está de más agregar paréntesis para hacer que las cosas sean más legibles aunque no sean estrictamente necesarios.

Detalles importantes

Precedencia

- La precedencia determina el orden en el que se llevan a cabo las operaciones en una *expresión*.
- La precedencia de operadores aritméticos sigue el orden PEMuDReS.
- Primero paréntesis, luego exponenciación, luego multiplicación y división, luego resta y suma.
- La exponenciación es asociativa hacia la derecha.

Detalles importantes

Precedencia

- La precedencia determina el orden en el que se llevan a cabo las operaciones en una *expresión*.
- La precedencia de operadores aritméticos sigue el orden PEMuDReS.
- Primero paréntesis, luego exponenciación, luego multiplicación y división, luego resta y suma.
- La exponenciación es asociativa hacia la derecha.

Un adelanto: Algunas funciones

Algunas funciones útiles

- `abs(x)`: Devuelve el valor absoluto.
- `complex(x,y)`: Devuelve el complejo $x + iy$
- `int(x)`: Devuelve la parte entera de x
- `float(n)`: Convierte el argumento en punto flotante
- `hex(n)`: Devuelve la representación hexadecimal del entero n
- `oct(n)`: Devuelve la representación octal de entero n
- `bin(n)`: Devuelve la representación binaria del entero n
- `round(n,[m])`: Redondea al flotante más cercano con m dígitos después de la coma o al entero más cercano si m está ausente.

Síntesis y recursos:

- Manual de referencia de Python
- Manual de la Librería estándar de Python