



Expresiones

Expresiones Numéricas

- Dada la falta de símbolos matemáticos en los teclados de la computadora, utilizamos el “lenguaje de la computadora” para expresar las operaciones matemáticas clásicas
- El asterisco es la multiplicación
- La potenciación (elevar a la potencia) tiene un aspecto diferente que en matemáticas

Operador	Operación
+	Suma
-	Resta
*	Multiplicación
/	División
**	Potencia
%	Resto

Expresiones Numéricas

```
>>> xx = 2
>>> xx = xx + 2
>>> print(xx)
4
>>> yy = 440 * 12
>>> print(yy)
5280
>>> zz = yy / 1000
>>> print(zz)
5.28
```

```
>>> jj = 23
>>> kk = jj % 5
>>> print(kk)
3
>>> print(4 ** 3)
64
```

4 R 3

5

23

20

3

Operador	Operación
+	Suma
-	Resta
*	Multiplicación
/	División
**	Potencia
%	Resto

Orden de Evaluación

- Cuando introducimos una cadena de operadores, Python debe saber cuál tiene que hacer primero
- Esto recibe el nombre de “precedencia del operador”
- Ahora, ¿qué operador “tiene precedencia” sobre los otros?

x = 1 + 2 * 3 - 4 / 5 ** 6

Reglas de Precedencia del Operador

De la regla de precedencia más alta a la regla de precedencia más baja:

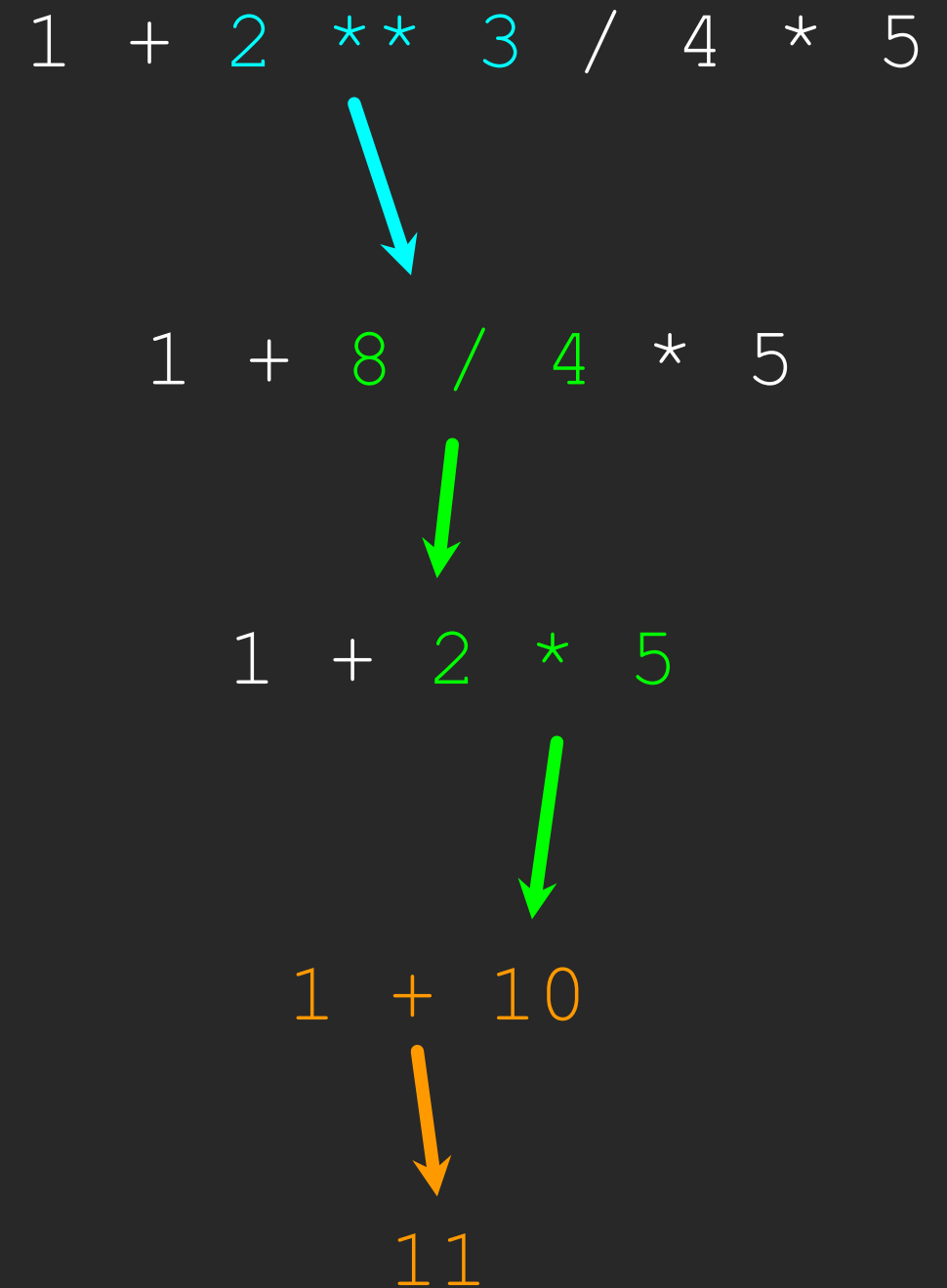

- Siempre se respetan los paréntesis
- Potenciación (elevar a la potencia)
- Multiplicación, división, resto
- Suma y resta
- Izquierda a derecha

Paréntesis
Potencia
Multiplicación
Suma
Izquierda a
derecha



```
>>> x = 1 + 2 ** 3 / 4 * 5
>>> print(x)
11.0
>>>
```

Paréntesis
Potencia
Multiplicación
Suma
Izquierda a
derecha



Precedencia del Operador

- Recuerde las reglas de arriba hacia abajo
- Cuando escribe un código, utilice paréntesis
- Cuando escribe un código, use las expresiones matemáticas más simples que le sea posible para que sean fáciles de entender
- Divida las series de operaciones matemáticas largas para que sean más claras

Paréntesis
Potencia
Multiplicación
Suma
Izquierda a
derecha



¿Qué Significa “Type” (Tipo)?

- En Python, las variables, literales y constantes tienen un “**type**” (tipo)
- Python sabe la **diferencia** entre un número entero y una cadena
- Por ejemplo “**+**” significa “suma” si se trata de número y “concatenación” si se trata de una cadena

```
>>> ddd = 1 + 4
>>> print(ddd)
5
>>> eee = 'hola ' + 'a
todos'
>>> print(eee)
Hola a todos
```

concatenación = unión

El “Type” (Tipo) Importa

- Python sabe cual es el “type” de todo
- Algunas operaciones están prohibidas
- No se puede “agregar 1” a una cadena
- Podemos preguntarle a Python de qué tipo se trata con la función `type()`

```
>>> eee = 'hola ' + 'a todos'
>>> eee = eee + 1
Trazas de rastreo (llamada más reciente a lo último): Archivo
"<stdin>", línea 1, in
<module>TypeError: Can't convert
'int' object to str implicitly
>>> type(eee)
<class 'str'>
>>> type('hola')
<class 'str'>
>>> type(1)
<class 'int'>
>>>
```

Diferentes Types (Tipos) de Número

- Los números tienen dos types (tipos)
 - **Enteros (int)**:
-14, -2, 0, 1, 100, 401233
 - **Números con punto flotante (float)**,
que tienen decimales: -2.5 , 0.0,
98.6, 14.0
- Hay otros tipos de números: son
variantes entre los números decimales
y los números enteros

```
>>> xx = 1
>>> type (xx)
<class 'int'>
>>> temp = 98.6
>>> type(temp)
<class 'float'>
>>> type(1)
<class 'int'>
>>> type(1.0)
<class 'float'>
>>>
```

Conversiones de Type (Tipo)

- Cuando introduce un número entero y un decimal en una expresión, el entero (int) se convierte **implícitamente** en uno decimal (float)
- Puede controlar esto con las funciones incorporadas `int()` y `float()`

```
>>> print(float(99) + 100)
199.0
>>> i = 42
>>> type(i)
<class 'int'>
>>> f = float(i)
>>> print(f)
42.0
>>> type(f)
<class 'float'>
>>>
```

División de Números Enteros

- La división de números enteros arroja un resultado con punto flotante

```
>>> print(10 / 2)
5.0
>>> print(9 / 2)
4.5
>>> print(99 / 100)
0.99
>>> print(10.0 / 2.0)
5.0
>>> print(99.0 / 100.0)
0.99
```

La división de enteros era diferente en
Python 2.x

Conversiones de Cadenas

- Puede también utilizar `int()` y `float()` para realizar conversiones entre cadenas y enteros
- Obtendrá un **error** si la cadena no contiene caracteres numéricos

```
>>> sval = '123'
>>> type(sval)
<class 'str'>
>>> print(sval + 1)
Trazas de rastreo (llamada más reciente a lo último): Archivo "<stdin>", línea 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
>>> ival = int(sval)
>>> type(ival)
<class 'int'>
>>> print(ival + 1)
124
>>> nsval = 'hola bob'
>>> niv = int(nsval)
Trazas de rastreo (llamada más reciente a lo último): Archivo "<stdin>", línea 1, in <module>
ValueError: invalid literal for int() with base 10: 'x'
```

Input (Entrada) del Usuario

- Podemos instruirle a Python que haga una pausa y lea los datos del usuario con la función `input()`
- La función `input()` regresa a la cadena

```
nam = input('Quién es usted')  
print('Bienvenido', nam)
```

Quién es usted

Chuck

Bienvenido Chuck

Crear un Programa