Identificadores y palabras clave

La primera regla se refiere a los personajes iniciales y de continuación.
El carácter de inicio puede ser cualquier cosa que Unicode
considere una letra, incluido el ASCII.

letras **(“a”, “b”, …, “z”, “A”, “B”, …, “Z”)**, el guión bajo **(“_”)**

“ Los identificadores son mayúsculas y minúscula sensible, por lo que, por ejemplo:

TAXRATE, Taxrate, TaxRate, taxRate y taxrate
son cinco diferentes identificadores.

Palabras clave de Python

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

¿Podemos decir si un identificador cae en una de estas categorías?
Python tiene un Función incorporada llamada **dir()** que devuelve una lista de los atributosde un objeto.
Si esto es llamado sin argumentos,
devuelve la lista de atributos integrados de Python.
Para ejemplo:

```
>>> dir(__ bui lti ns__) ['Arit hme tic Error', 'AssertionError',
 'Attri but eEr ror', ... 'sum', 'super', 'tuple', 'type', 'vars', 'zip']
```

Un solo guión bajo puede usarse como identificador,
y dentro de un intérprete interactivo o Python Shell
Por ejemplo:

```
>>>for _ in (0, 1, 2, 3, 4, 5): print( " Hel lo")
```

Módulo Matemático Ejemplo:

```
>>>π = math.pi
 ε = 0.0000001
 nueva_área = π  radio  radio
if abs(nu eva _área - vieja_ área) < ε:
print("las áreas han conver gid o")
```

Ventana Python Shell de IDLE ejemplo:

Identificadores y palabras clave (cont)

```
>>> stretc h-f actor = 1 Syntax Error: can't assign to operator (...)
>>> 2miles = 2 Syntax Error: invalid syntax (...)
>>> str = 3 # Legal but BAD
 >>> l'impôt31 = 4 Syntax Error: EOL while scanning single -quoted string (...)
 >>> l_impôt31 = 5 >>>
```

Un identificador de Python válido es una secuencia no vacía de caracteres de cualquier longitud.
que consta de un “carácter de inicio” y cero o más “caracteres de continuación”.
Un identificador de este tipo debe obedecer un par de reglas y seguir ciertas convenciones.

Strings

Las cadenas están representadas por el tipo de datos str inmutable que contiene una secuencia de caracteres Unicode.. La función **str()** también se puede usar como función de conversión.

Ejemplos:

a = "Las 'comillas' simples están bien; se debe utilizar el carácter de escape \"dobles\".".
b = 'Las \'comillas\' simples deben tener carácter de escape; Los "dobles" están bien.'

Comparando cadenas

Las cadenas admiten los operadores de comparación habituales <, <=, ==, !=, > y >=. Estos Los operadores comparan cadenas byte a byte en la memoria. Por ejemplo, el punto de carácter 0x00C5) se puede representar en bytes codificados en UTF8 en tres diferentes formas: [0xE2, 0x84, 0xAB], [0xC3, 0x85] y [0x41, 0xCC, 0x8A].

Tipos integrales

Python proporciona dos tipos integrales integrados, int y bool.

El tamaño de un número entero está limitado
únicamente por la memoria de la máquina.
Hay dos objetos booleanos integrados:
Verdadero y Falso.

Enteros Ejemplo:

```
>>> 14600926 # decimal
 14600926
>>> 0b1101 111 011 001 010 110 11110 # binary
14600926
 >>> 0o67545336 # octal
14600926
>>> 0xDECADE # hexade cimal
14600926
```

Booleanos Ejemplo:

Tipos integrales (cont)

```
>>> t = True
>>> f = False
>>> t and f False
>>> t and True True
```

Tipos de punto flotante

Python proporciona tres tipos de valores de punto flotante:

Los tipos flotantes

Complejos integrados

El tipo decimal

Decimal de la biblioteca estándar.

Números de punto flotante Ejemplo:

```
def equal_ flo at(a, b):
 return abs(a - b) <= sys.fl oat _in fo.e psilon




s = 14.25.h ex() # str s == '0x1.c 800 000 000 000p+3'
f = float.f ro mhex(s) # float f == 14.25
 t = f.hex() # str t == '0x1.c 800 000 000 000p+3'
```

Números complejos Ejemplo:

```
>>> z = -89.5+2.125j
 >>> z.real, z.imag (-89.5, 2.125)



>>> z.conj ugate() (-89.5-2.125j)
 >>> 3-4j.c onj ugate() (3+4j)
```

Numeros decimales Ejemplo:

```
>>> import decimal
 >>> a = decimal.Decimal(9876)
>>> b = decimal.Decimal("54321.012345678987654321")
 >>> a + b
Decimal('64197.012345678987654321')



>>> 23 / 1.05
21.904761904761905
 >>> print(23 / 1.05)
21.9047619048
>>> print( dec ima l.D eci mal(23) / decimal.Decimal("1.05"))
 21.904 761 904 761 904 761 904 76190
```