Capítulo 1-5
El lenguaje Python

1. Introducción

Ahora que los tipos de datos no tienen secretos, es hora de entrar en más detalles sobre el lenguaje Python en sí.

2. La función print()

La primera función que vamos a estudiar es, por supuesto, la función `print()`.

Observación

Desde la versión 3 de Python es una función. Antes era una instrucción, por lo que no era necesario paréntesis.

De manera predeterminada, `print()` muestra el objeto solicitado y luego envía un final de línea (*Line Feed*).

Este comportamiento se puede modificar con la opción `end`, que le permite especificar lo que desea al final.

```
>>> for x in [1,2,3]:
...   print(x , end="-")
...
1-2-3->>> # sin fin de línea ... desde '-' en lugar
```

`print()` puede mostrar también varios valores de golpe.

```
>>> print(1,2,3)
1 2 3
```

`print()` permite modificar el separador en este caso.

```
>>> print(1,2,3, sep="/")
1/2/3
```

Pero sobre todo, `print()` permite formatear los datos de dos maneras:

Primero llegó el formateo C-STYLE, al igual que `printf()` en lenguaje C o el comando `printf` del shell.

Ejemplo

```
>>> print( "[ %10s ]" % "hello")
[      hello ]
```

Luego vino el formateo con el comando `format()`, que permite generar documentos de una manera simple y elegante.

```
>>> print( "[ {} al {} ]".format( "hola", "mundo" ))
[ hola al mundo ]
```

2.1 Print() formateado C-STYLE

Este método de formateo casi desapareció del lenguaje Python, pero los módulos grandes como el módulo 'logging' todavía lo usan y nuevos tipos de datos, como la clase '`bytes`', finalmente lo usarán.

```
"<cadena con formato>" % valor
o
"<cadena con múltiples formatos>" % (valor, valor, ...)
```

Y como a muchos, nos resulta bastante fácil escribir a máquina, además de que algunos hábitos son difíciles de romper.

© Editions ENI - All rights reserved

La expresión del formato:

– El carácter '%' indica el inicio del formato
– Una opción (# 0 <espacio> - +)
– Una clave (opcional) entre paréntesis (nombre_de_campo)
– Una indicación de longitud mínima (opcional)
– Una indicación opcional formada por un punto ('.'), seguida de un número, que se utiliza para definir la precisión después del punto decimal de los números y la longitud máxima de las cadenas de caracteres.
– Un conversor de formato (s f x X c% para los principales).

```
>>> "%05.2f" % 3.14159265359
'03.14'
```

`%`	marcador de inicio de formato
`0`	opción que permite llenar los espacios vacíos de la izquierda con ceros
`5`	indicación de longitud
`.2`	indicador de precisión, por lo tanto, dos dígitos después del punto decimal
`f`	formato de conversión: flotante

Lo mismo con una clave y un diccionario:

```
>>> "%(pi)05.2f" % {'pi':3.14159265359}
'03.14'
```

Para una cadena de caracteres:

```
>>> "%2.2s" % "ABCD"
'AB'
```

Aquí están, como ejemplos, los principales formatos generalmente utilizados.

Ejemplo con cadenas de caracteres

```
>>> "[%s]" % "Hola compañeros"
'[Hola compañeros]'

## formateo a la izquierda con la opción -
>>> "%-10.10s" % "AAA"
```

```
'AAA       '

## si no, es a la derecha
>>> "%10.10s" % "AAA"
'       AAA'

>>> "%05d | %20s | %20s |" % ( 12, "Nombre", "Apellidos")
'00012 |                  Nombre |               Apellidos |'
```

Ejemplos con claves

```
>>> print("%(clave)s %(clave)s %(clave)s" % {"clave": "Valor"})
Valor Valor Valor
```

```
# archivo: print_1.py

contactos = [
       { "nombre": "Dark", "apellido": "Vador" },
       { "nombre": "Luke", "apellido": "Skywalker" },
       { "nombre": "Han", "apellido": "Solo" },
       ]

for num, contacto in enumerate(contactos):
   contacto['num'] = num
   print( "%(num)05d | %(apellido)20s | %(nombre)20s |" % contacto)
```

La ejecución da:

```
$ python print_1.py

00000 |                Vador |                 Dark |
00001 |            Skywalker |                 Luke |
00002 |                 Solo |                  Han |
```

Otros ejemplos particulares (observe que esto puede servir más adelante):

`%r`	usa la función `repr()` de un objeto.
`%a`	usa la función `ascii()` de un objeto.

© Editions ENI - All rights reserved

2.2 Print() formateo cadena.format()

Después de los conceptos básicos de impresión a la antigua, ahora necesitamos estudiar con más detalle el método `format()` de la clase String.

Este al final reemplazará al otro.

La sintaxis es muy similar:

```
"<cadena con formateo>".format(valor )
ou
"<cadena con múltiples formateos>" .format(valor, valor, ... )
```

Ejemplo

```
>>> print( "[ {} the {} ]".format( "Hola", "mundo" ))
[ Hola al mundo ]
```

Pero con este método, casi llegamos a un mini lenguaje de especificación de formato en el lenguaje Python.

Aquí está la fórmula simplificada:

{	Inicio de formato
Clave	Clave del valor para mostrar (opcional)
:	Separador de clave/formato
carácter de relleno	Cualquier carácter (opcional)
alineación	Tenga en cuenta que debe especificarse la alineación. "<" ">" "^" izquierdo derecho centro (opcional) '=' solo para números
signo	Para números con signo +/- o espacio (opcional)
Indicador de longitud	Número entero opcional
.precisión	Una indicación opcional formada por un punto ('.') seguido de un número, que se utiliza para definir la precisión después del punto decimal de los números y la longitud máxima de las cadenas de caracteres.
tipo	Tipo de conversión
}	Fin del formato

Y las opciones adicionales:

#	Para forzar la alternativa.
0	Forzar el signo antes del carácter de relleno. +00010 por ejemplo
Opción de agrupación	Guion bajo ('_') o coma (",") opcional para el formato de los números.

Tipo de conversión	Descripción

b	Binario: muestra las cifras en base 2
	`>>> print( "{:b}".format(255))`
	`11111111`

c	Carácter: convierte los enteros en unicode
	`>>> print( "{:c}".format(65))`
	A
	`>>> print( "{:c}".format(128521))`
	☺

d	Entero decimal
	`>>> print( "{:3d}".format(300))`
	`300`

e o E	Visualización en notación de forma exponencial con precisión predeterminada de 6
	`>>> print( "{:e}".format(3.1416))`
	`3.141600e+00`

f o F	Flotante
	`>>> print( "{:3.2f}".format(3.1416))`
	`3.14`

g o G	Flotante
	`>>> print( "{:g}".format(3.1416))`
	`3.1416`

© Editions ENI - All rights reserved

Tipo de conversión	Descripción

o	Byte
	`>>> print( "{:o}".format(255))`
	`377`

s	Cadena de caracteres (formato por defecto)
	`>>> print( "[{:-^30.30s}]".format (" Hello "))`
	`[----------- Hello ------------]`

x ou X	Hexadecimal
	`>>> print( "{:2x}".format(255))`
	`ff`

%	Porcentaje: multiplica el número por 100 y usa el formato 'f'
	`>>> print( "{:2.2%}".format(.75))`
	`75.00%`
	`>>> print( "{:2.2%}".format(1.25))`
	`125.00%`

La técnica Printf/C-STYLE permitiá transmitir un diccionario como argumento, `format()` también lo permite.

```
#archivo: print3.py

contactos = [
       { "nombre": "Dark", "apellido": "Vador" },
       { "nombre": "Luke", "apellido": "Skywalker" },
       { "nombre": "Han", "apellido": "Solo" },
       ]

for num, contacto in enumerate(contactos):
   print( "{num:05d}|{contacto[nombre]:15s}|{contacto[apellido]:15s}"
.format( num=num, contacto=contacto))
```

La forma especial de expresar argumentos para `format()` es la forma de pasar argumentos en Python; esto se estudiará con las funciones.