Listas de números enteros: el tipo range ()

En Python 3, range es un tipo de datos. El tipo range es una **lista inmutable de números enteros en sucesión aritmética**.

Listas

Las listas (list) son un tipo de datos muy flexible, que se comenta en la lección listas. Como para ver los valores de un range se necesita convertirlo a una lista, se comenta aquí la definición de lista, sin entrar en más detalles.

Una lista es un conjunto ordenado de elementos del mismo o diferente tipo, cuyo contenido puede modificarse.

Se representan escribiendo los elementos entre corchetes y separados por comas.

Las variables de tipo lista hacen referencia a la lista completa.

```
>>> lista = [1, "abcde", 45.5, -32]
>>> lista
[1, 'abcde', 45.5, -32]
```

Una lista que no contiene ningún elemento se denomina **lista vacía**:

```
>>> lista = [ ]
>>> lista
[ ]
```

El tipo range

El tipo range es una **lista inmutable de números enteros en sucesión aritmética**.

- Inmutable significa que, a diferencia de las listas, los range no se pueden modificar.
- Una sucesión artimética es una sucesión en la que la diferencia entre dos términos consecutivos es siempre la misma.

Un range se crea llamando al tipo de datos con uno, dos o tres argumentos numéricos, como si fuera una función.

Nota: En Python 2, range() se consideraba una función, pero en Python 3 no se considera una función, sino un tipo de datos , aunque se utiliza como si fuera una función.

El tipo range() con un único argumento se escribe range(n) y crea una lista inmutable de *n* números enteros consecutivos que empieza en 0 y acaba en *n - 1*.

Para ver los valores del range(), es necesario convertirlo a lista mediante la función list ().

```
>>> x = range(10)
>>> x
range(0, 10)
>>> list(x)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(7)
range(0, 7)
>>> list(range(7))
[0, 1, 2, 3, 4, 5, 6]
```

Si *n* no es positivo, se crea un range vacío.

```
>>> list(range(-2))
[]
>>> list(range(0))
[]
```

El tipo range con dos argumentos se escribe range(m, n) y crea una lista inmutable de enteros consecutivos que empieza en *m* y acaba en *n - 1*.

```
>>> list(range(5, 10))
[5, 6, 7, 8, 9]
>>> list(range(-5, 1))
[-5, -4, -3, -2, -1, 0]
```

Si *n* es menor o igual que *m*, se crea un range vacío.

```
>>> list(range(5, 1))
[]
>>> list(range(3, 3))
[]
```

El tipo range con tres argumentos se escribe range(m, n, p) y crea una lista inmutable de enteros que empieza en *m* y acaba justo antes de superar o igualar a *n*, aumentando los valores de *p* en *p*. Si *p* es negativo, los valores van disminuyendo de *p* en *p*.

```
>>> list(range(5, 21, 3))
[5, 8, 11, 14, 17, 20]
>>> list(range(10, 0, -2))
[10, 8, 6, 4, 2]
```

El valor de *p* no puede ser cero:

```
>>> range(4,18,0)
Traceback (most recent call last):/span>
  File "<pyshell#0>", line 1, in <module>
    range(4,18,0)
ValueError: range() arg 3 must not be zero
```

Si *p* es positivo y *n* menor o igual que *m*, o si *p* es negativo y *n* mayor o igual que *m*, se crea un range vacío.

```
>>> list(range(25, 20, 2))
[]
>>> list(range(20, 25, -2))
[]
```

En los range(*m*, *n*, *p*), se pueden escribir *p* range distintos que generan el mismo resultado. Por ejemplo:

```
>>> list(range(10, 20, 3))
[10, 13, 16, 19]
>>> list(range(10, 21, 3))
[10, 13, 16, 19]
>>> list(range(10, 22, 3))
[10, 13, 16, 19]
```

En resumen, los tres argumentos del tipo range(m, n, p) son:

- *m*: el valor inicial
- *n*: el valor final (que no se alcanza nunca)
- *p*: el paso (la cantidad que se avanza cada vez).

Si se escriben sólo dos argumentos, Python le asigna a *p* el valor 1. Es decir range(m, n) es lo mismo que range(m, n, 1)

Si se escribe sólo un argumento, Python, le asigna a *m* el valor 0 y a *p* el valor 1. Es decir range(n) es lo mismo que range(0, n, 1)

El tipo range() sólo admite argumentos enteros. Si se utilizan argumentos decimales, se produce un errror:

```
>>> range(3.5, 10, 2)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    range(3.5, 10, 2)
TypeError: range() integer start argument expected, got float.
```

Nota: En versiones muy antiguas de Python se podían utilizar argumentos decimales, que Python truncaba a enteros.

Concatenar range()

No se pueden concatenar tipos range(), ya que el resultado de la concatenación puede no ser un tipo range()).

```
>>> range(3) + range(5)
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    range(3) + range(5)
TypeError: unsupported operand type(s) for +: 'range' and 'range'
```

Pero sí se pueden concatenar tipos range() previamente convertidos en listas. El resultado es lógicamente una lista, que no se puede convertir a tipo range().

```
>>> list(range(3)) + list(range(5))
[0, 1, 2, 0, 1, 2, 3, 4]
```

No se pueden concatenar tipos range(), ni aunque el resultado sea una lista de números enteros en sucesión aritmética.

```
>>> range(1, 3) + range(3, 5)
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    range(3) + range(5)
TypeError: unsupported operand type(s) for +: 'range' and 'range'
>>> list(range(1, 3)) + list(range(3, 5))
[1, 2, 3, 4]
```

La función len()

La función len() devuelve la longitud de una cadena de caracteres o el número de elementos de una lista. El argumento de la función len() es la lista o cadena que queremos "medir".

```
>>> len("mensaje secreto")
15
>>> len(["a","b","c"])
3
>>> len(range(1, 100, 7))
15
```

El valor devuelto por la función len() se puede usar como parámetro de range().

```
>>> list(range(len("mensaje secreto")))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
>>> list(range(len(["a","b","c"])))
[0, 1, 2]
>>> list(range(len(range(1, 100, 7))))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```

Última modificación de esta página: 20 de febrero de 2017