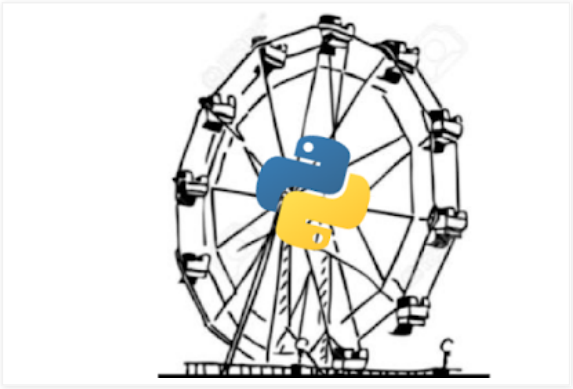
Python 3 para impacientes

"Simple es mejor que complejo" (Tim Peters)

miércoles, 26 de agosto de 2015

Bucles eficientes con Itertools



Itertools es un módulo de la librería estándar de Python que incorpora funciones que devuelven objetos iterables, es decir, estructuras de datos basadas en elementos que pueden recorrerse secuencialmente y que pueden utilizarse en procesos repetitivos (bucles).

Estas funciones fueron diseñadas para una ejecución rápida, haciendo un uso eficiente de la memoria, con la idea de resolver algoritmos basados en bucles más complicados que aquellos que habitualmente se suelen implementar en un programa para recorrer los elementos de una lista, diccionario, etc.

Funciones que devuelven iterables infinitos

Agrupa un conjunto de funciones **Itertools** que devuelven un iterable que no se interrumpirá si no se fuerza un final, por ejemplo, cuando se cumpla una determinada condición.

count()

Devuelve un objeto iterable en la que el primer elemento tendrá el valor inicial (***start***) y los sucesivos se irán incrementando/decrementado con el valor del paso (***step***), de manera ininterrumpida.

itertools.count(start=0 [, step=1])

En el siguiente ejemplo cuando se alcanza un valor determinado se fuerza la interrupción del bucle:

```
from itertools import *
for valor in count(5, 3):
    print(valor, end = ' ')
    if valor == 20: break
```

Salida: 5 8 11 14 17 20

cycle()

Devuelve un objeto iterable con los elementos (de principio a fin) del iterable de entrada, que se reproducirán una y otra vez mientras no se fuerce un final.

itertools.cycle(iterable)

A continuación, varios ejemplos:

cycle() con una cadena:

Buscar

Buscar

Python para impacientes

Python
IPython
EasyGUI
Tkinter
JupyterLab
Numpy

Anexos

Guía urgente de MySQL
Guía rápida de SQLite3

Entradas + populares

Dar color a las salidas en la consola
En Python para dar color a las salidas en la consola (o en la terminal de texto) existen varias posibilidades. Hay un método basado ...

Instalación de Python, paso a paso
Instalación de Python 3.6 A finales de 2016 se produjo el lanzamiento de Python 3.6 . El propósito de esta entrada es mostrar, pas...

Añadir, consultar, modificar y suprimir elementos en Numpy
Acceder a los elementos de un array. [], [,], ... Acceder a un elemento de un array. Para acceder a un elemento se utiliz...

Variables de control en Tkinter
Variables de control Las variables de control son objetos especiales que se asocian a los widgets para almacenar sus valore...

Cálculo con arrays Numpy
Numpy ofrece todo lo necesario para obtener un buen rendimiento cuando se trata de hacer cálculos con arrays. Por como está concebid...

Operaciones con fechas y horas. Calendarios
Los módulos datetime y calendar amplían las posibilidades del módulo time que provee funciones para manipular expresiones de ti...

Tkinter: interfaces gráficas en Python
Introducción Con Python hay muchas posibilidades para programar una interfaz gráfica de usuario (GUI) pero Tkinter es fácil d...

Convertir, copiar, ordenar, unir y dividir arrays Numpy
Esta entrada trata sobre algunos métodos que se utilizan en Numpy para convertir listas en arrays y viceversa; para copiar arrays d...

Threading: programación con hilos (I)
En programación, la técnica que permite que una aplicación ejecute simultáneamente varias operaciones en el mismo espacio de proceso se...

El módulo random
El módulo random de la librería estándar de Python incluye un conjunto de funciones

```
contador = 0
for elemento in cycle("Python"):
    print(elemento, end = ' ')
    contador += 1
    if contador == 12: break
```

Salida: P y t h o n P y t h o n

cycle() con una lista:

```
contador = 0
for elemento in cycle([10, 12, 14]):
    print(elemento, end = ' ')
    contador += 1
    if contador == 5: break
```

Salida: 10 12 14 10 12

cycle() con un diccionario:

```
contador = 0
for elemento in cycle({'x':1, 'y':2, 'z': 3}):
    print(elemento, end = ' ')
    contador += 1
    if contador == 9: break
```

Salida: x z y x z y x z y

repeat()

Devuelve el objeto completo, una y otra vez, de manera indefinida a menos que se especifique el número de veces (***times***) que hay que ejecutar el bucle.

itertools.repeat(object[, times])

A continuación, varios ejemplos:

repeat() con un entero:

```
for elemento in repeat(3, 5):
    print(elemento, end = ' ')
```

Salida: 3 3 3 3 3

repeat() con map():

```
print(list(map(pow, range(5), repeat(3))))
```

Salida: [0, 1, 8, 27, 64]

Funciones que devuelven iterables que finalizan

Agrupa funciones del módulo **Itertools** que devuelven iterables que finalizan con la secuencia de entrada más corta.

accumulate()

La función devuelve un iterable con sumas acumuladas o totales acumulados derivados de los resultados obtenidos al aplicar alguna función (***func***).

Los elementos del objeto iterable se evalúan tomando el primer elemento con el segundo; después el resultado obtenido de ambos elementos con el tercero y así sucesivamente.

itertools.accumulate(iterable[, func])

La función que se incluya debe tener dos argumentos (uno por cada elemento que se evalúa) y los elementos de la entrada iterable, necesariamente, serán del tipo que acepte dicha función. Además, si la entrada iterable está vacía la salida también lo estará.

A continuación, varios ejemplos:

accumulate() con la función implícita que acumula sumas:

que permiten obtener de distintos modos números a...

Archivo

agosto 2015 (2)

python.org

pypi.org

Sitios

- ActivePython
- Anaconda
- Bpython
- Django
- Flask
- Ipython
- IronPython
- Matplotlib
- MicroPython
- Numpy
- Pandas
- Pillow
- PortablePython
- PyBrain
- PyCharm
- PyDev
- PyGame
- Pypi
- PyPy
- Pyramid
- Python.org
- PyTorch
- SciPy.org
- Spyder
- Tensorflow
- TurboGears

```
for acumulado in accumulate([1, 2, 3, 4, 5]):
    print(acumulado, end = ' ')
```

Salida: 1 3 6 10 15

accumulate() con función max para 'acumular' el valor máximo:

```
for valor_maximo in accumulate([1, 3, 2, 5, 4], max):
    print(valor_maximo, end = ' ')
```

Salida: 1 3 3 5 5

acumulate() con función lambda:

```
for diferencia in accumulate([10, 30, 50], lambda a, b: b-a):
    print(diferencia, end = ' ')
```

Salida: 10 20 30

chain()

La función devuelve un iterable construido con todos los elementos de los objetos iterables de entrada.

itertools.chain(*iterables)

```
for elemento in chain([1, 2], [3, 4, 5]):
    print(elemento ** 2, end = ' ')
```

Salida: 1 4 9 16 25

chain.from_iterable()

Es similar a **chain()** pero solo admite un argumento iterable.

classmethod chain.from_iterable(iterable)

```
for elemento in chain.from_iterable(["un","dos","tres"]):
    print(elemento, end = ' ')
```

Salida: u n d o s t r e s

compress()

La función devuelve un iterable a partir de los elementos de **data** que tienen cada uno un valor asociado en **selectors** que actúan como un filtro. Todos los elementos con el valor ***True*** o ***1*** serán los utilizados para construir el iterable. El proceso finaliza cuando se agoten los elementos en **data** o en **selectors**.

itertools.compress(data, selectors)

```
for elemento in compress("iterable", [1,0,1,0,1,0,0,1]):
    print(elemento, end = ' ')
```

Salida: i e a e

dropwhile()

Construye un iterable a partir del iterable de entrada sin devolver ningún elemento hasta que la condición del **predicado** sea falsa. Después de ese momento se devuelven todos los elementos que resten.

dropwhile(predicate, iterable)

```
for elemento in dropwhile(lambda valor: valor == 'x',
                          ['x','x','y','z','x','x']):
    print(elemento, end = ' ')
```

Salida: y z x x

filterfalse()

Esta función devuelve un iterable con los elementos del iterable de entrada que no cumplan la condición expresada en el **predicado**. Si el predicado es **None** devuelve los elementos con valor **False** o **0**.

itertools.filterfalse(predicate, iterable)

```
for elemento in filterfalse(lambda valor: valor == 'x',
                            ['x','x','y','z','x','x']):
    print(elemento, end = ' ')
```

Salida: y z

groupby()

Devuelve un iterable con los elementos del iterable de entrada agrupados por el dato utilizado como clave (**key**).

itertools.groupby(iterable, key=None)

A continuación, varios ejemplos:

groupby() con lista ordenada:

En el siguiente ejemplo la lista de tuplas contiene ciudades de diferentes países ordenados alfabéticamente:

```
ciudades = [("Bolivia", "Sucre"), ("Bolivia", "La Paz"),
            ("Chile", "Valdivia"), ("Chile", "Arica"),
            ("España", "Cádiz"), ("Perú", "Cusco"),
            ("Perú", "Lima")]

for clave, grupo in groupby(ciudades, lambda x: x[0]):
    print(clave, list(grupo))
```

Salida:
Bolivia [('Bolivia', 'Sucre'), ('Bolivia', 'La Paz')]
Chile [('Chile', 'Valdivia'), ('Chile', 'Arica')]
España [('España', 'Cádiz')]
Perú [('Perú', 'Cusco'), ('Perú', 'Lima')]

groupby() y itemgetter() con lista desordenada:

En el siguiente ejemplo la lista de tuplas contiene ciudades de diferentes países que no están ordenados alfabéticamente.

Para que se pueda agrupar la información de cada país es necesario ordenar la lista. Para ello se utiliza la función **sorted()** con la función **itemgetter()** del módulo **operator**. Esta función permite establecer el criterio de ordenación. En el ejemplo, como cada tupla contiene dos elementos (país, ciudad) **itemgetter(0)** establece que el criterio de orden será el primer elemento de la tupla, es decir, el país.

```
from operator import itemgetter
ciudades = [("Perú", "Cusco"), ("Chile", "Valdivia"),
            ("Bolivia", "Sucre"), ("Bolivia", "La Paz"),
            ("España", "Cádiz"), ("Chile", "Arica"),
            ("Perú", "Lima")]

ciudades = sorted(ciudades, key=itemgetter(0))
for clave, grupo in groupby(ciudades, itemgetter(0)):
    print(clave, list(grupo))
```

Salida:
Bolivia [('Bolivia', 'Sucre'), ('Bolivia', 'La Paz')]

```
Chile [('Chile', 'Valdivia'), ('Chile', 'Arica')]
España [('España', 'Cádiz')]
Perú [('Perú', 'Cusco'), ('Perú', 'Lima')]
```

islice()

La función devuelve un iterable con una selección de elementos del iterable de entrada. Permite retornar un número de elementos partiendo desde el inicio del iterable o un rango específico.

itertools.islice(iterable, stop)

A continuación, algunos ejemplos:

Devuelve elementos partiendo desde el comienzo:

```
for elemento in islice("KLMNOPQRST", 5):
    print(elemento, end = ' ')
```

Salida: K L M N O

itertools.islice(iterable, start, stop [, step])

Devuelve los elementos que hay desde una posición inicial a una final:

```
for elemento in islice("KLMNOPQRST", 5, 7):
    print(elemento, end = ' ')
```

Salida: P Q

Devuelve los elementos que hay desde una posición inicial a una final, separados entre sí por un valor fijo de elementos:

```
for elemento in islice("1234567890", 0, 8, 2):
    print(elemento, end = ' ')
```

Salida: 1 3 5 7

starmap()

Construye un objeto iterable aplicando una función que utiliza como argumentos los elementos del iterable de entrada, devolviendo una secuencia con los resultados obtenidos.

itertools.starmap(function, iterable)

En el ejemplo se recorre una lista de tuplas y se construye el iterable a devolver con el valor más alto de los elementos que hay en cada una de ellas:

```
for elemento in starmap(max,
                        [(10,2),(2,32),(63,54),(4,45)]):
    print(elemento, end = ' ')
```

Salida: 10 32 63 45

takewhile()

Construye un iterable a partir del iterable de entrada devolviendo elementos mientras la condición del predicado es verdadera. En el momento que cambie a falsa no devolverá más elementos aunque exista alguno que cumpla la condición.

itertools.takewhile(predicate, iterable)

En el ejemplo se recorre una lista de cadenas comprobando si su longitud es igual a 1:

```
for elemento in takewhile(lambda x: len(x) == 1,
                          ['a','b','ab','bc','c']):
    print(elemento, end = ' ')
```

Salida: a b

tee()

Devuelve varios iterables independientes (por defecto, 2) sobre la base de una sola entrada original.

itertools.tee(iterable, n=2)

```
a, b = tee([1,2,3,4])
for elemento1, elemento2 in zip(a, b):
    print("a:", elemento1)
    print("b:", elemento2)
```

Salida:
a: 1
b: 1
a: 2
b: 2
a: 3
b: 3

zip_longest()

Devuelve un iterable que agrega elementos de cada uno de los iterables de entrada. Si los iterables de entrada tienen distinta longitud se completará utilizando el valor de **fillvalue**, hasta que se alcance el final del iterable que tenga una longitud mayor.

itertools.zip_longest(*iterables, fillvalue=None)

```
for elemento in zip_longest(['x','y','z'], ['0','1'],
                            fillvalue='#'):
    print(elemento, end = ' ')
```

Salida: ('x', '0') ('y', '1') ('z', '#')

Generadores para combinatoria

Agrupa una serie de funciones que generan iterables como resultado de operaciones de combinatoria en las que se utilizan otros iterables de entrada.

product()

Realiza el producto cartesiano con los elementos de los iterables de entrada devolviendo un iterable basado en tuplas con el resultado.

itertools.product(*iterables, repeat=1)

```
for elemento in product("XYZ", "mn"):
    print(elemento, end = ' ')
```

Salida: ('X', 'm') ('X', 'n') ('Y', 'm') ('Y', 'n') ('Z', 'm') ('Z', 'n')

```
for elemento in product("X", repeat = 4):
    print(elemento, end = ' ')
```

Salida: ('X', 'X', 'X', 'X')

```
for elemento in product("XYZ", repeat = 2):
    print(elemento, end = ' ')
```

Salida: ('X', 'X') ('X', 'Y') ('X', 'Z') ('Y', 'X') ('Y', 'Y') ('Y', 'Z') ('Z', 'X') ('Z', 'Y') ('Z', 'Z')

permutations()

Devuelve un objeto iterable basado en tuplas con permutaciones de longitud **r** a partir de los elementos del objeto de entrada.

itertools.permutations(iterable, r=None)

```
for elemento in permutations("123", 2):
    print(elemento, end = ' ')
```

Salida: ('1', '2') ('1', '3') ('2', '1') ('2', '3') ('3', '1') ('3', '2')

combinations()

Devuelve un objeto iterable basado en tuplas con las combinaciones sin repetición posibles, de una longitud **r**, a partir de los elementos del objeto de entrada.

itertools.combinations(iterable, r)

```
for elemento in combinations("123", 2):
    print(elemento, end = ' ')
```

Salida: ('1', '2') ('1', '3') ('2', '3')

combinations_with_replacement()

Devuelve un objeto iterable basado en tuplas con las combinaciones con repetición posibles, de una longitud **r**, a partir de los elementos del objeto de entrada.

itertools.combinations_with_replacement(iterable, r)

```
for elemento in combinations_with_replacement("123", 2):
    print(elemento, end = ' ')
```

Salida: ('1', '1') ('1', '2') ('1', '3') ('2', '2') ('2', '3') ('3', '3')

Ir al índice del tutorial de Python

Publicado por Pherkad en 15:52

Etiquetas: Python3

Entrada más reciente Inicio Entrada antigua