Python 3 para impacientes

"Simple es mejor que complejo" (Tim Peters)

domingo, 20 de septiembre de 2015

El módulo random

El módulo **random** de la librería estándar de Python incluye un conjunto de funciones que permiten obtener de distintos modos números aleatorios o, para ser rigurosos, pseudoaleatorios.

¿Por qué números pseudoaleatorios y no aleatorios?

Las funciones a las que hacemos referencia, cada vez que son invocadas, devuelven un valor de una secuencia de números predeterminada. Esta secuencia tiene un periodo bastante largo, es decir, es necesario obtener muchos números antes de que se vuelva a reproducir la misma secuencia. De ahí, el tratamiento de pseudo (o falso), aunque la utilidad que nos ofrezca sea equivalente al de los números aleatorios.

Los generadores de números aleatorios verdaderos se basan en procesos naturales que evolucionan de manera impredecible. En ellos, aunque se pueda tener una idea global de lo que puede suceder, tener la certeza absoluta de lo que ocurrirá en cada momento es imposible y por ello las secuencias de valores que generan tienen una bajísima probabilidad de repetirse.

Las aplicaciones que tienen los números aleatorios o pseudoaleatorios son muchas: juegos de azar, animaciones de videojuegos, simulaciones, análisis de datos, pruebas de algoritmos, etc.

randint()

La función **randint()** devuelve un número entero incluido entre los valores indicados. Los valores de los límites inferior y superior también pueden aparecer entre los valores devueltos.

En el siguiente ejemplo se realizan cinco "sorteos" y se obtiene, para cada uno de ellos, un regalo de diez posibles.

```
import random

regalos = ['sartén', 'jamón', 'mp4', 'muñeca', 'tv',
           'patín', 'balón', 'reloj', 'bicicleta', 'anillo']

for sorteo in range(5):
    regalo = regalos[random.randint(0, 9)]
    print('Sorteo', sorteo + 1, ':', regalo)
```

A continuación, en otro ejemplo, se obtienen diez números enteros entre los que pueden aparecer números negativos, positivos y el cero.

```
for numero in range(10):
    print(random.randint(-3, 3))
```

randrange()

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...

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

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...

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...

Tkinter: Tipos de ventanas
Ventanas de aplicación y de diálogos En la entrada anterior tratamos los distintos gestores de geometría que se utilizan para di...

Threading: programación con hilos (I)
En programación, la técnica que permite que una aplicación ejecute

La función **randrange()** devuelve enteros que van desde un valor inicial a otro final separados entre sí un número de valores determinados. Esta separación (o paso) se utiliza en primer lugar con el valor inicial para calcular el siguiente valor y los sucesivos hasta llegar al valor final o al más cercano posible.

En el ejemplo que sigue se obtienen 25 números que van desde el 3 al 16. Como el paso es 3 nunca se obtendría el valor 16.

```
print('\nValores posibles: 3, 6, 9, 12, 15')
for i in range(25):
    print(random.randrange(3, 16, 3), end=' ')
```

A continuación, se utiliza la forma abreviada para obtener números que van desde el 0 a cualquier número (en este caso el 4) con el paso 1.

```
print('\nvalores posibles: 0, 1, 2, 3')
for i in range(25):
    print(random.randrange(4), end=' ')
```

random()

La función **random()** devuelve un número ***float*** entre 0 y 1

En el ejemplo se obtienen tres números con muchos decimales entre 0 y 1.

```
for numero in range(3):
    print(random.random(), end=' ')
```

uniform()

La función **uniform()** devuelve un número ***float*** incluido entre los valores indicados.

```
for numero in range(3):
    print(random.uniform(100, 105), end=' ')
```

seed()

Cuando interese obtener varias veces la misma secuencia de números pseudoaleatoria se puede utilizar la función **seed()** que fija mediante una "semilla" el mismo comienzo en cada secuencia, permitiendo con ello obtener series con los mismos valores.

A continuación, se muestra un ejemplo donde se realizan dos series de cinco sorteos; y en cada serie se obtienen los mismos regalos y en el mismo orden.

La semilla en este caso se fija con un valor numérico (0 en este caso) pero también se puede utilizar una cadena o una unidad de tiempo obtenida con la función **time()** del módulo **time**; o incluso puede expresarse con cualquier objeto "hashable" de Python.

```
regalos = ['sartén', 'jamón', 'mp4', 'muñeca', 'tv',
           'patín', 'balón', 'reloj', 'bicicleta', 'anillo']

for serie in range(2):
    print('\nserie:', serie + 1)
    random.seed(0)
    for sorteo in range(5):
        regalo = regalos[random.randint(0, 9)]
        print('Sorteo', sorteo + 1, ':', regalo)
```

choice()

La función **choice()** se utiliza para seleccionar elementos al azar de una lista.

En el siguiente ejemplo se obtiene un elemento de los cinco existentes en una lista.

```
transporte = ['bici', 'moto', 'coche', 'tren', 'avión']
print('Hoy viajaré en:', random.choice(transporte))
```

suffle()

La función **shuffle()** 'mezcla' o cambia aleatoriamente el orden de los elementos de una lista antes de realizar la selección de alguno de ellos. Esta mezcla recuerda en el caso de los juegos de cartas la acción de barajar un número de veces antes de repartir o seleccionar cartas.

El el siguiente ejemplo se realizan dos 'mezclas' antes de que se obtenga el elemento.

```
lista = ['rojo', 'verde', 'amarillo']

random.shuffle(lista)
print('mezcla1', lista)

random.shuffle(lista)
print('mezcla2', lista)
print(lista[random.randint(0,2)])
```

sample()

La función **sample()** devuelve de una lista de elementos un determinado número de elementos diferentes elegidos al azar.

En el ejemplo que sigue **sample()** devuelve dos elementos de una lista que contiene tres.

```
lista = ['x', 'y', 'z']
print(random.sample(lista, 2))
```

getrandbits()

De una longitud de bits determinada la función **getrandbits()** devuelve números enteros decimales que se puedan representar con los mismos.

Con 1 bit se pueden representar 2 números (0,1); con 2 bits, cuatro (0,1,2,3); con 3 bits, ocho (0,1,2,3,4,5,6,7) y con n-bits, 2 elevado a n.

A continuación se generan 10 números decimales que se pueden representar con 3 bits.

```
for num in range(10):
    print(random.getrandbits(3), end=' ')
```

getstate() y setstate()

La función **getstate()** obtiene el estado del generador de números en un momento dado. Dicho estado se puede utilizar con posterioridad para reiniciar con **setstate()** el generador a ese estado anterior y controlar así la secuencia.

En el siguiente ejemplo se generan números aleatorios en dos series de 3 números cada una que se van alternando. Cada vez que la primera serie se encuentra en el segundo número (serie1 = 1) se obtiene el estado del generador que es reiniciado con posterioridad cuando la segunda serie se encuentra en el segundo número (serie2 = 1). Este proceso se repite 20 veces.

```
contador = 0
random.seed(0)
while True:
    contador += 1
    if contador > 20:
        break
    for serie1 in range(3):
        if serie1 == 1:
            estado = random.getstate()
        print('serie1', serie1, random.random())

    for serie2 in range(3):
        if serie2 == 1:
            random.setstate(estado)
        print('serie2', serie2, random.random())
```

Otra posibilidad a considerar consiste en salvar el estado a un fichero binario para cuando interese, leerlo para fijar el generador y continuar con la secuencia de ese momento anterior.

```
import pickle

# Salvar estado

with open('estado.dat', 'wb') as fich:
```

```
    pickle.dump(random.getstate(), fich)

...

# Leer estado

with open('estado.dat', 'rb') as fich:
    estado = pickle.load(fich)

...

# Establecer estado al momento salvado

random.setstate(estado)
```

La clase Random

La clase **Random** permite instanciar varios generadores de números aleatorios "trabajando" cada uno con su propia secuencia sin interferir en el resto, o bien, si se inician con la misma semilla generando la misma secuencia.

En el ejemplo siguiente, en la primera parte, se instancian dos generadores por separado y se generan 3 números aleatorios, con cada uno de ellos, obteniéndose secuencias diferentes. En la segunda parte, se utiliza la semilla de la cadena "Python" para iniciar los dos generadores y producir así números basados en la misma secuencia.

```
print('Inicializa dos generadores:')

serie1 = random.Random()
serie2 = random.Random()

for num in range(3):
    print(serie1.random(), serie2.random())

print('Se utiliza la misma semilla:')

semilla = 'Python'
serie1 = random.Random(semilla)
serie2 = random.Random(semilla)

for num in range(3):
    print(serie1.random(), serie2.random())
```

La clase random.SystemRandom

La clase **random.SystemRandom** utiliza la función **os.urandom()** para generar números aleatorios a partir de fuentes proporcionadas por el sistema operativo y no está disponible en todos los sistemas.

Su funcionamiento no se basa en el estado del software y las secuencias numéricas no son reproducibles. Por tanto, el método **seed()** no tiene ningún efecto y es ignorado. Además, los métodos **getstate()** y **setstate()** no se han implementado para esta clase y producirán una excepción si son invocados.

El siguiente ejemplo muestra cómo generar números aleatorio con la clase **random.SystemRandom**.

```
serie = random.SystemRandom()

for num in range(3):
    print(serie.random())
```

Otras funciones disponibles

Las siguientes funciones generan secuencias de números aleatorios no uniformes (mediante funciones estadísticas) para usos más especializados:

triangular(), betavariate(), expovariate(), gammavariate(), gauss(), lognormvariate(), normalvariate(), vonmisesvariate(), paretovariate() y weibullvariate()

Consultar: https://docs.python.org/3.5/library/random.html

Ir al índice del tutorial de Python

Publicado por Pherkad en 13:41

Etiquetas: Python3

Entrada más reciente Inicio Entrada antigua