Curso Práctico de Python: Creación de un CRUD

Artículo

Iterators and generators

David Aroesti 17 de Octubre de 2018

Aunque no lo sepas, probablemente ya utilices iterators en tu vida diaria como programador de Python. Un iterator es simplemente un objeto que cumple con los requisitos del Iteration Protocol (protocolo de iteración) y por lo tanto puede ser utilizado en ciclos. Por ejemplo,

```
for i in range(10):
    print(i)
```

En este caso, la función range es un iterable que regresa un nuevo valor en cada ciclo. Para crear un objeto que sea un iterable, y por lo tanto, implemente el protocolo de iteración, debemos hacer tres cosas:

- Crear una clase que implemente los métodos **iter** y **next**
- **iter** debe regresar el objeto sobre el cual se iterará
- **next** debe regresar el siguiente valor y aventar la excepción StopIteration cuando ya no hayan elementos sobre los cual iterar.

Por su parte, los generators son simplemente una forma rápida de crear iterables sin la necesidad de declarar una clase que implemente el protocolo de iteración. Para crear un generator simplemente declaramos una función y utilizamos el keyword yield en vez de return para regresar el siguiente valor en una iteración. Por ejemplo,

```
def fibonacci(max):
    a, b = 0, 1
    while a < max:
        yield a
        a, b = b, a+b
```

s importante recalcar que una vez que se ha agotado un generator ya no odemos utlizarlo y debemos crear una nueva instancia. Por ejemplo,

```
fib1 = fibonacci(20)
fib_nums = [num for num in fib1]
...
double_fib_nums = [num * 2 for num in fib1] # no va a funcionar
double_fib_nums = [num * 2 for num in fibonacci(30)] # sí funciona
```

Escribe aquí tu pregunta

+ 2

wilantury Estudiante · hace 18 días

```
>>> deffibonacci(max):
...     a, b=1, 1
... while a < max :
... yield a
...             a, b = b, a+b
...
>>> fibonacci(20)
<generator object fibonacci at 0x7f2d85c580f8>
>>> im = fibonacci(20)
>>> fib_nums = [num for num in im]
>>> fib_nums
[1, 1, 2, 3, 5, 8, 13]
>>> fib_nums = [num for num in im]
>>> fib_nums
[] # tengo como respuesta un objeto vacio, pues una vez usado el generator se debe instanciar de nuevo.
```

1

1

Guillermo Vara De Gante Estudiante • hace 2 meses

¿Este tipo de sentencias es exclusiva de python?. Me refiero a los generadores y a la funciones yield?

1

Jose Manuel Salazar • hace 6 días

No, Tambien existe en JS [generators] (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Iterators_and_Generators)

1

Adrián Andrés Sosa Estudiante • hace 2 meses

Well, very useful

1

Murray Walker Greer Cifuentes Estudiante • hace 3 meses

Interesante...

1

Osvaldo Trejo Estudiante • hace 4 meses

muy buena explicación!

1

David Behar Estudiante • hace 4 meses

En el curso profesional de javascript explican muy bien que son los generadores.
https://platzi.com/clases/1642-javascript-profesional/22173-generato-2/

5

alvamar zapata Estudiante • hace 4 meses

Uso de listas

1

Esau Rosales Estudiante • hace 5 meses

Cuando escribes return en una función, acaba en esa linea, pero yield no termina una función, sino que continúa su ejecución.
Mas información...

3

thejb Estudiante • hace 7 meses

Que significa que se ah agotado un generador?

1

karencarolina • hace 7 meses

que ya no tiene mas elementos que retornar

2

Facundo Nicolás García Martoni • hace 6 meses

¡Hola @thejb! Significa que tienes que crear una instancia nueva del generador ya que el mismo no se reiniciará

2

Ver más respuestas

thejb Estudiante • hace 7 meses

Alguien me ayuda a leer esta linea?

```
a, b = b, a+b
```

El resultado seria este?

```
a = b
b = a+b
```

u.mitchel • hace 7 meses

En Python existe la assignation multiple, lo que corresponde a asignar valores a muchas variables en una sola linea de código, por eso, si tenemos:
a , b = 1 , 2
Lo que enverado estamos diciendo es: A la variable a le asignas el valor 1, y a la variable b le asignas el valor 2.
Entendemos entonces, que la variable se le asigna el valor en la posición que se encuentre la variable, y las posiciones las entendemos por la separación de comas.

5

David Santiago Pinchao Ortiz Estudiante • hace 8 meses

El Yield entonces es como una especie de return ? Digamos retorna el valor de donde fue invocada ? pero sigue la ejecucion

2

Facundo Nicolás García Martoni • hace 6 meses

¡Hola @SanGeeky! Estás en lo correcto, es la palabra clave que indica el momento de retorno de un valor en un generador 😉 Te dejo este link de un artículo que puede servirte para profundizar y entender mejor el tema.

1

jmsuarezsuncion Estudiante • hace 9 meses

def fibonacci(max):
a, b = 0, 1
while a < max:
yield a#es para retornar valores que se iteran. Todo esto guardado en un generator
a, b = b, a+b
fibonacci(20)#se crea un generador
#<generator object fibonacci at 0x000001C2269DE570>
for i in fibonacci(20):#para leer sus valores
print(i)#Se imprimen la secuencia fibonacci

1

Isaac Alejandro Pimentel Morales Estudiante • hace 9 meses

Hola, el codigo que aparece en el texto:

```
        yield a
        a, b = b, a+b
```

Es equivalente a este que tal vez les puede ser mas claro para entender:

```
def fibonacci(max):
  a = 0
  b = 1

  while a < max:
    yield a
    aux = a
    a = b
    b = aux + b
```

yield lo que hace es ir regresando el valor de a cada vez que el código pasa por el, la sintaxis que usan en el código del articulo “a, b = b, a+b” es una manera de asignar valores a las variables pero mantenido el valor de las variables ahorrándote usar el aux que use en el otro código, ya que fibonacci requiere sumar los dos números anteriores para generar el siguiente, les recomiendo que si no saben como funciona la serie de fibonacci la investiguen y después lean de nuevo ambos códigos y entenderán como funcionan, si tiene alguna duda pueden preguntarme y con gusto contestare

3

Manuel Mosquera Estudiante • hace 9 meses

No queda muy claro que digamos.

1

Facundo Nicolás García Martoni • hace 6 meses

¡Hola Manuel! ¿Qué fue lo que no entendiste? Te dejo dos artículos muy buenos que explican a fondo el tema:

- Iterators
- Generators

1

sergio-medina93 Estudiante • hace 10 meses

Un poco confuso jejej

Facundo Nicolás García Martoni • hace 6 meses

¡Hola Sergio! ¿Qué fue lo que no entendiste? Te dejo dos artículos muy buenos que explican a fondo el tema:

- Iterators
- Generators

1

sjbarrientos Estudiante • hace 11 meses

Gracias

1

lucasezequiel Estudiante • hace 11 meses

Esto si que no lo entiendo así explicado. Debo investigar en otro lado.

6

David Daniel Acerbo • hace 10 meses

Hola tal vez te sea de ayuda, https://www.youtube.com/watch?v=TLVnoqXGWpY&list=PLU8oAlHdN5BlvPxziopYZRd55pdqFwkeS&index=19

6

Facundo Nicolás García Martoni • hace 6 meses

¡Hola Lucas! ¿Qué fue lo que no entendiste? Te dejo dos artículos muy buenos que explican a fondo el tema:

- Iterators
- Generators

1

Ver más respuestas

CHRISTIAN PATRICK LAVADO Estudiante • el año pasado

Gracias

1

Uso de listas

1. https://pythonista.io/events/eventos/news/iteradores-y-generadores-en-python-3
2. http://buildingskills.itmaybeahack.com/book/python-2.6/html/p02/p02c08_generators.html

4

ignaciom8 Estudiante • el año pasado

Cuando ejecuto el código desde el directorio desde un cmd, donde antes voy entrando a cada carpeta, corre perfectamente y no se cierra. Pero a la hora de hacerlo con doble click al archivo, me permite ejecutarlo, pero al finalizar la operación ya sea create o cualquier otra se cierra al instante.

1

Facundo Nicolás García Martoni • hace 6 meses

¡Hola Ignacio! Esto puede ser un problema con Windows. Te recomiendo probar Pyinstaller para generar un **.exe** de tu programa 😉

1

Victor Hugo Soria Estudiante • el año pasado

Gracias!

1

Jhon Henry Bernal Rodriguez Estudiante • el año pasado

No entiendo este punto a, b = 0, 1

2

Jcion • el año pasado

Puede ser que sea equivalente a lo siguiente

```
a = 0
b = 1
```

En `a, b = 0, 1` capaz se están creando dos variables en la misma línea.

6

Uso de listas

Puedes leer esto para entender la serie de fibonacci - una función que escriba la serie de Fibonacci hasta un límite determinado:
a=0 ; b= 1

- http://docs.python.org.ar/tutorial/3/controlflow.html#definiendo-funciones

1

Ver más respuestas

Jhon Henry Bernal Rodriguez Estudiante • el año pasado

n

1

Carlos Oblitas Villegas Estudiante • el año pasado

Porque en el ultimo dice que no va a funcionar con fib1 y con fibonacci(30), si funcionara??

```
double_fib_nums = [num * 2fornumin fib1] # no va a funcionar
double_fib_nums = [num * 2fornumin fibonacci(30)] # sí funciona
```

1

Julián Andrés Santos Méndez • el año pasado

Porque la instancia fib1 ya fue usada y los Iterators solo se pueden usar una vez.
Toca crear otra instancia para que funcione.

1

Nicolas Humberto Sulca Vega Estudiante • el año pasado

otras formas de implentar fibonacci

1

jramosaguas Estudiante • el año pasado

Excelente. Cada generators es un iteratos, pero no al revés.

No me quedo muy claro el uso!

1

Daniel García • el año pasado

Te entiendo estoy igual pero solo es cuestion de repasarlo

1

Ana Lima • el año pasado

Daniel, a lo que se refiere el profesor es que un iterator es simplemente un objeto que cumple ciertos requisitos y **puede ser utilizado en ciclos.**

2

Facundo Nicolás García Martoni • hace 6 meses

¡Hola Daniel! ¿Qué fue lo que no entendiste? Te dejo dos artículos muy buenos que explican a fondo el tema:

- Iterators
- Generators

1

Ver más respuestas

willeonardo19 Estudiante • el año pasado

Hola dejo mi aporte. espero que sea un poco mas legible y les sirva de ayuda.

```
#
deffibonacci(max):
  #max = max
  print(max)
  print('')
  a, b = 0, 1
  while a < max:
    yield a
    print(a)#el valor de retorno
    a, b = b, a+b


fib1 = fibonacci(20)
fib_nums = [num for num in fib1]
print(fib_nums)#se muestra la lista generada
```

Uso de listas

```
print(double_fib_nums)#se muestra una lista vacia por que ya no exiten los valores
de fib1

double_fib_nums = [num * 2for num in fibonacci(30)] # sí funciona, ya que se vulve a
llamar a fibonacci(x), un un leve cambio que n se multiplicara x 2
print(double_fib_nums)#se muestran los valores recien generados #
print('')
print('Ejemplos extra'.center(50,'-'))
##ejemplos
# define a list
lista = [4, 7, 0, 3]

# get an iterator using iter()
mi_iterador = iter(lista)#establecemos el objeto iterador

## iterate through it using next()

#prints 4
print(next(mi_iterador))#muestra el siguiente elemento de la lista

#prints 7
print(next(mi_iterador))#muestra el siguiente elemento de la lista

## next(obj) is same as obj.__next__()

#prints 0
print(mi_iterador.__next__())#muestra el siguiente elemento de la lista

#prints 3
print(mi_iterador.__next__())#muestra el siguiente elemento de la lista

## This will raise error, no items left
next(mi_iterador)#muestra error xq ya no hay elementos a mostrar
```

Iterator: Un objeto tipo iterator es un objeto que representa un flujo de datos, el cual puede ser recorrido en un proceso iterativo, como un bucle for, dentro de una función map o filter, en la creación de una list comprehension o generador, o en una comparación in.
Todo objeto iterator contiene implementado un método **next**() que es llamado en cada iteración devolviendo los sucesivos elementos del flujo de datos cada vez. El flujo de datos del objeto no tiene por qué estar guardado en memoria, sino que puede ser generado en tiempo real en cada iteración.
El objeto iterator guarda un estado interno para saber cuál fue el último elemento obtenido. Así, en la siguiente llamada a **next**(), se obtendrá el siguiente elemento correcto.
Cuando ya no quedan más elementos en el flujo de datos del iterator, la función **next**() lanza StopIteration. El estado interno no se reinicia automáticamente al llegar al final del flujo o al empezar a recorrerlo de nuevo. Es decir, sólo se puede recorrer una vez.
Además, tiene implementado el método **iter**() que devuelve el propio objeto iterator. Esto es necesario para poder implementar bucles con objetos iterator, como explicaremos después.

Uso de listas

thejb • hace 7 meses

Perdon, alguien me puede explicar esta linea?
“fib_nums = [num for num in fib1]”

1

Facundo Nicolás García Martoni • hace 7 meses

@thejb lo que observas en esa línea se denomina **list comprehension** y es una manera simplificada del lenguaje para crear los elementos de una lista sin necesidad de usar un ciclo. Te dejo el enlace de la clase correspondiente, que está más adelante en el curso: https://platzi.com/clases/1378-python-practico/14172-python-comprehensions/

1

Ver más respuestas

Jesús Ignacio García Fernández Estudiante • el año pasado

el codigo con algunos print para ver el funcionamiento

```
deffibonacci(max):
    print(max)
    a, b = 0, 1

    while a < max:
        yield a
        a, b = b, a+b


if __name__ == '__main__':
    print("inicio")
    fib1 = fibonacci(20)

    fib_nums = [num for num in fib1]
    print(fib_nums)

    double_fib_nums = [num * 2for num in fib1] # no va a funcionar
    print(double_fib_nums)
    double_fib_nums = [num * 2for num in fibonacci(30)] # sí funciona
    print(double_fib_nums)
    print("fin")
```

5

Algunos ejemplos de una clase que implemente los métodos iter y next:
Construye tu propio iterado en python.

25

Roberto Sobarzo • el año pasado

Gracias!!

3

Sylar017 • el año pasado

Gracias !!

1

lpal248 • el año pasado

Gracias!!!

1

daniel-medina137 • el año pasado

Gracias por el aporte

2

Nestor Cepeda • el año pasado

Un placer!

1

Eduardo Cecilio Flores Ambrosio • el año pasado

buena esa

1

Marcos Andres Garcia Carreño • el año pasado

yes

grax

1

Neiro13 • el año pasado

Gracias por tu aporte

1

CHRISTIAN PATRICK LAVADO • el año pasado

Gracias

1

Ver más respuestas