Генераторы python

Суханов А.Я.

Генераторы

- Генератор создает объект, который используется для итерирования, при этом он генерирует по одному элементу и не хранит их в памяти после итерирования. Фактически сгенерировав объект, происходит переход к следующему, предыдущие в памяти не хранятся.

```
generator = (i for i in range(5))
print(generator)
for item in generator:
    print(item)
```

Yield. Функция выполняется до первого yield, генерирует объект и при следующем исполнении начинает с того места, где завершился последний выполненный yield

```
def fgenerator(n):
    print("start")
    for i in range(n):
        print("in cycle")
        yield i
    yield 100
    print("between")
    yield 200
    print("end")

p = fgenerator(5)

for i in p:
    print(i)
print(list(p))
```

```
start
in cycle
0
in cycle
1
in cycle
2
in cycle
3
in cycle
4
100
between
200
end
[]
```

Меняем местами, видим, что после перебора генератора, элементов в нем для перебора не остается

```
def fgenerator(n):
    print("start")
    for i in range(n):
        print("in cycle")
        yield i
    yield 100
    print("between")
    yield 200
    print("end")

p = fgenerator(5)

print(list(p))
for i in p:
    print(i)
```

```
start
in cycle
in cycle
in cycle
in cycle
in cycle
in cycle
between
end
[0, 1, 2, 3, 4, 100, 200]
>>>
```

```
def fgenerator(n):
    print("start")
    for i in range(n):
        if(i>5):
            return
        print("in cycle")
        yield i
    yield 100
    print("between")
    yield 200
    print("end")

p = fgenerator(4)

print(list(p))
for i in p:
    print(i)
```

```
start
in cycle
in cycle
in cycle
in cycle
in cycle
between
end
[0, 1, 2, 3, 100, 200]
```

```
def fgenerator(n):
    print("start")
    for i in range(n):
        if(i>5):
            return
        print("in cycle")
        yield i
    yield 100
    print("between")
    yield 200
    print("end")

p = fgenerator(7)

print(list(p))
for i in p:
    print(i)
```

```
start
in cycle
in cycle
in cycle
in cycle
in cycle
in cycle
[0, 1, 2, 3, 4, 5]
>>>
```

Пример генерации случайных чисел больше 0.5

```
from random import *

def frandom(n):
    i=0
    while i<n:
        x = random()
        if x>0.5:
            i=i+1
            yield x

print(len(list(frandom(100))))
print(list(frandom(100)))
```

Конечно, можно 0.5+random()*0.5, но как пример для возможного создания более сложных генерируемых объектов.

Пример генерации случайного гладкого сигнала

```
from math import *
from random import *
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import UnivariateSpline
def func(n):
  nw = int(n/(random()*10+2))
  w = (np.random.rand(nw)-0.5)*2
  x = np.linspace(-1,1,nw)
  spl  =  UnivariateSpline(x=x,y=w, k=3 , s=1)
  xs =  np.linspace(-1,1,n)
  ws = spl(xs)
  return ws
def funcgener(size,n):
  for i in range(size):
    yield func(n)
```

```
n = 100
res = [it for it in
funcgener(10,n)]
print(len(res))
x = np.linspace(-1,1,n)
y = func(n)
err = np.random.randn(n)
yerr = y+err*0.1*y
plt.plot(x,y)
plt.plot(x,yerr)
plt.show()
```