Politechnika Świętokrzyska

Laboratorium

Programowanie w języku Python 2

Ćwiczenie 2

Programowanie obiektowe
Iteratory i generatory

Cel ćwiczenia

Celem ćwiczenia jest zapoznanie studentów z narzędziami programistycznymi i bibliotekami języka Python.

dr inż Robert Kazała

Indeksowanie tablic

Indeksowanie klasyczne z range.
W Pythonie 3 funkcja zakresu zachowuje inaczej niż w Pythonie 2.

```
from pylab import *
a=eye(10)
b=zeros(a.shape)
for w in range(a.shape[0]):
    for k in range(a.shape[1]):
        print(w,k)
        b[w,k]=a[w,k]

imshow(b)
```

Indeksowanie klasyczne z xrange
W Pythonie 3 nie ma xrange, ale funkcja zakresu zachowuje się jak xrange w Pythonie 2.

```
a=zeros((10,8))
a[:,(3,5)]=1
b=zeros(a.shape)
for w in xrange(a.shape[0]):
    for k in xrange(a.shape[1]):
        print(w,k)
        b[w,k]=a[w,k]

imshow(b,interpolation='none',cmap='Blues')
```

Wykorzystanie comprehension list

```
a=zeros((10,8))
a[0:-1:2,:]=1
b=zeros(a.shape)
for (y,x) in [(y,x) for y in range(a.shape[0]) for x in
range(a.shape[1]) if a[y,x]>0.5]:
    print(y,x)
    b[y,x]=a[y,x]

imshow(b,interpolation='none',cmap=cm.gray)
```

Wykorzystanie iteratora ndenumerate

```
a=zeros((11,9))
a[::2,::2]=1
b=zeros(a.shape)
for (y,x),i in ndenumerate(a):
    print(y,x,i)
    b[y,x]=a[y,x]
```

```
imshow(b,interpolation='none',cmap=cm.autumn)
```

Wykorzystanie iteratora ndindex

```
a=zeros((11,9))
a[5::3,0:8:3]=1
b=zeros(a.shape)
for y,x in ndindex(a.shape):
    print y,x
    b[y,x]=a[y,x]

imshow(b,interpolation='bilinear',cmap=cm.hot)
```

Iteratory

Iterator to obiekt zawierający policzalną liczbę wartości. Można po nim iterować, co oznacza, że można przechodzić przez wszystkie wartości.
Technicznie rzecz biorąc, w Pythonie iterator to obiekt, który implementuje protokół iteratora, który składa się z metod __iter__() and __next__().

Przykład

Zwraca iterator z krotki i wypisuje każdą wartość:

```
mytuple = ("apple", "banana", "cherry")
myit = iter(mytuple)

print(next(myit))
print(next(myit))
print(next(myit)
```

Tworzenie iteratorów

Utwórz iterator, który zwraca liczby, zaczynając od 1, a każda sekwencja wzrośnie o dwa (zwracając 1,3,5 itd.)

```
class MyNumbers:
  def __iter__(self):
    self.a = 1
    return self

  def __next__(self):
    x = self.a
    self.a += 2
    return x

myclass = MyNumbers()
myiter = iter(myclass)

print(next(myiter))
```

Przykład

Utwórz iterator, który zwraca liczby, zaczynając od 1, a każda sekwencja wzrośnie o dwa (zwracając 1,3,5 itd.), Stop po 20 iteracjach:

```
class MyNumbers:
  def __iter__(self):
    self.a = 1
    return self

  def __next__(self):
    if self.a <= 20:
      x = self.a
      self.a += 2
      return x
    else:
      raise StopIteration

myclass = MyNumbers()
myiter = iter(myclass)

for x in myiter:
  print(x)
```

Przykład

Obliczanie ciągu Fibonacciego

```
class Fib:
    def __init__(self, max):
       self.max = max

def __iter__(self):
    self.a = 0
    self.b = 1
    return self

def __next__(self):
    fib = self.a
    if fib > self.max:
        raise StopIteration
    self.a, self.b = self.b, self.a + self.b
    return fib
```

Generatory

Generatory to proste i wydajne narzędzie do tworzenia iteratorów. Są napisane jak zwykłe funkcje, ale używają instrukcji yield, kiedy chcą zwrócić dane. Za każdym razem, gdy wywoływana jest

funkcja next (), generator wznawia od miejsca, w którym został przerwany (zapamiętuje wszystkie wartości danych i ostatnią instrukcję).
Wszystko, co można zrobić za pomocą generatorów, można również wykonać za pomocą iteratorów opartych na klasach. Generatory są tak kompaktowe, ponieważ metody __iter __ () i __next __ () są tworzone automatycznie.
Inną kluczową cechą jest to, że zmienne lokalne i stan wykonania są automatycznie zapisywane między wywołaniami. Dzięki temu funkcja jest łatwiejsza do napisania i znacznie bardziej przejrzysta niż podejście wykorzystujące zmienne instancji, takie jak self.index i self.data.
Oprócz automatycznego tworzenia metod i zapisywania stanu programu, gdy generatory zakończą się, automatycznie generują StopIteration.

Przykład generatora:

```
def reverse(data):
    for index in range(len(data)-1, -1, -1):
        yield data[index]

for char in reverse('golf'):
    print(char)
```

Literatura

Zadania

1. Uruchomić i przeanalizować działanie wszystkich przykładów z instrukcji.
2. Na własnych przykładach zaprezentować wykorzystanie różnych metod iterowania tablic.
3. Na własnych przykładach zaprezentować działanie iteratorów.
4. Na własnych przykładach zaprezentować działanie generatorów.
5. Wykorzystując pętle i iteratory utworzyć tablicę o wymiarach 100x100x100 o elementach losowych. Posortować elementy w każdej warstwie tablicy w kierunku przesuniętym o 90 stopni. Zamienić co drugą warstwę i co drugą płaszczyznę pionową na 0. Ustawić wewnętrzne powierzchnie sześcianów na wartość od 0 do 9 zaczynając od zera. Wyświetlić przekroje przez tablicę pokazujące działanie algorytmu.
6. Wykorzystując pętle i funkcje wbudowane utworzyć tablicę o wymiarach 100x100x100 o elementach losowych. Posortować elementy w każdej warstwie tablicy w kierunku przesuniętym o 90 stopni. Zamienić co drugą warstwę i co drugą płaszczyznę pionową na 0. Ustawić wewnętrzne powierzchnie sześcianów na wartość od 0 do 9 zaczynając od zera. Wyświetlić przekroje przez tablicę pokazujące działanie algorytmu.
7. Wykorzystując iteratory, generatory i funkcje wbudowane utworzyć tablicę o wymiarach 100x100x100 o elementach losowych. Posortować elementy w każdej warstwie tablicy w kierunku przesuniętym o 90 stopni. Zamienić co drugą warstwę i co drugą płaszczyznę pionową na 0. Ustawić wewnętrzne powierzchnie sześcianów na wartość od 0 do 9 zaczynając od zera. Wyświetlić przekroje przez tablicę pokazujące działanie algorytmu.
8. Wykorzystując operatory zakresu i funkcje wbudowane (nie wolno wykorzystać pętli, iteratora i generatora) utworzyć tablicę o wymiarach 100x100x100 o elementach losowych.

Posortować elementy w każdej warstwie tablicy w kierunku przesuniętym o 90 stopni. Zamienić co drugą warstwę i co drugą płaszczyznę pionową na 0. Ustawić wewnętrzne powierzchnie sześcianów na wartość od 0 do 9 zaczynając od zera. Wyświetlić przekroje przez tablicę pokazujące działanie algorytmu.