Wykład 7

- Iterator
- Generatory
- Funkcje generujące
- Wyrażenia generujące
- Generatory rekurencyjne

1

Iterator

Iterator to obiekt zawierający policzalną liczbę wartości po którym można iterować, co oznacza, że można przechodzić przez wszystkie wartości. W Pythonie iterator to obiekt, który zawiera metody __iter__() i __next__().

Przykład:

```
lista = ["ala","ola", "ula"]
myit = iter(lista)

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

2

Tworzenie Iteratora

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

  def __next__(self):
    x = self.a
    self.a += 1
    return x

myclass = MyNumbers()
myiter = iter(myclass)

print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter))
```

3

Generatory

```
def podz(n):
  "Funkcja generująca podzielniki właściwe liczby n"
  for p in range(1,n):
    if n%p==0:
      yield p
# end def


for i in podz(120):
  print(i)


l = [ x for x in podz(24) ]
print(l)
```

4

Generator a funkcja generująca

```
>>> def podz(n):
>>>   for p in range(1,n):
>>>     if n%p==0:
>>>       yield p


>>> g=podz(24)
>>> print(g)
<generator object podz at 0x00000258A72A20C8>

>>> next(g)
1
>>> next(g)
2
>>> g.__next__()
3
```

5

Wyrażenie generujące

```
>>> n = 24
>>> g = (p for p in range(1,n) if n%p==0)
>>> g
<generator object <genexpr> at 0x000001DD225620C8>
>>> next(g)
1
>>> next(g)
2
>>> next(g)
3
>>> next(g)
4
>>> next(g)
6
>>> next(g)
8
```

6

Generator nieskończony

```
def fib():
  "Funkcja generująca kolejne wyrazy ciągu Fibonacciego"
  a,b = 1,1
  while True:
    yield a
    a,b = b,a+b
# end



suma = 0
for w in fib():
  print(w)
  suma += w
  if suma>10000:
    break
```

7

Liczby pierwsze

```
def primes():
  "Funkcja generuje kolejne liczby pierwsze"
  lp = [2]
  p = 2
  yield p
  while True:
    p += 1
    for d in lp:
      if p%d==0: break
    else:
      lp.append(p)
      yield p
# end
```

8

Własności liczb bliźniaczych

- Liczba 5 jest bliźniacza zarówno z 3, jak i z 7,
- Największe znane dziś liczby bliźniacze, każda składająca się z 200 700 cyfr, to $2996863034895 \cdot 2^{1290000} \pm 1$ znalezione w 2016 roku,
- Za wyjątkiem par 3 i 5 oraz 5 i 7, ostatnimi cyframi liczb bliźniaczych mogą być: 1 i 3 (na przykład 11 i 13), 7 i 9 (na przykład 17 i 19) oraz 9 i 1 (na przykład 29 i 31).
- Dla każdej pary liczb bliźniaczych większych lub równych 5, liczba naturalna między nimi (rozdzielająca parę) jest podzielna przez 6. Wynika to z faktu, że jest ona parzysta i ponieważ w każdej trójce kolejnych liczb jest liczba podzielna przez trzy (bo dwie pozostałe są pierwsze), to mamy również podzielność przez iloczyn liczb dwa i trzy.

Wikipedia

9

Liczby bliźniacze

```
def twins():
  """Funkcja generuje kolejne liczby bliźniacze"""

  g = primes()
  w1 = next(g)
  while True:
    w2 = next(g)
    if w2-w1==2:
      yield w1,w2
      w1 = w2
# end

for a,b in twins():
  print(a,b)
```

10

Generator rekurencyjny

```
def gen(n):
  if n==0:
    yield ""
  else:
    g = gen(n-1)
    while True:
      try:
        c = next(g)
        yield c+'0'
        yield c+'1'
      except StopIteration:
        break
# end def


for s in gen(4):
  print(s)
```

```
def gen(n):
  if n==0:
    yield ""
  else:
    for c in gen(n-1):
      yield c+'0'
      yield c+'1'
# end def



for s in gen(4):
  print(s)
```

11

Permutacje

```
def perm(s):
  "Funkcja generuje permutacje liter w napisie s"
  if len(s)==1:
    yield s
  else:
    for p in perm(s[:-1]):
      for i in range(len(s)):
        yield p[:i]+s[-1]+p[i:]
# end def

for a in perm('ABCD'):
  print(a)
```

12

Kombinacje

```
def komb(s,k):
  "Funkcja generuje k-elementowe kombinacje z napisu s"

  if k==1:
    for x in s: yield x
  elif len(s)==k:
      yield s
  else:
    for x in komb(s[1:],k): yield x
    for x in komb(s[1:],k-1): yield s[0]+x

# end


for a in komb('ABCDE',3):
  print(a)
```

13

Wariacje bez powtórzeń

```
def war(s,k):
  """Funkcja generuje k-elementowe wariacje
     bez powtórzeń z napisu s"""

  for ko in komb(s,k):
    for p in perm(ko):
      yield p
# end def


for w in war('ABCDE',3):
  print(w)
```

14

Podzbiory

```
def podzb(s):
 "Funkcja generuje podzbiory ze zbioru liter s"

  if len(s)==0:
    yield ""
  else:
    for x in podzb(s[1:]):
      yield x
      yield s[0]+x
# end

for a in podzb('ABCD'):
  print(a)
```

15