Предговор

08.11.2022

За какво си говорихме предния път?

- За жената на Жорката.
- За АЛО АЛО!
- За Изключения.
- За `with`

Питанка

```
try:
   print([3, 4, 5][6])
except KeyError as e:
   print('I seem to have misplaced my keys')
   print(type(e))
except Exception as e:
   print('Oh man, oh man, oh man…')
   print(type(e))
except IndexError as e:
   print('It tends to happen with indexable stuff, bummer though.')
   print(type(e))
except:
   print('I have no idea what just happened!')
```

```
'Oh man, oh man, oh man…' <class 'IndexError'>
```

Питанка №2 @Йоан питанка за with?

08. Итератори и Генератори

8 ноември 2022

Що е итератор?

- Когато имаме някакви данни, обикновено искаме да ги обхождаме.
- Как итерираме? - обикновено с `for` loop.
- Примери за итеруеми
  - `List`
  - `Set`
  - `Dict`
  - `Tuple`
  - `Map & filter` създават итеруеми обекти
  - Всичко в collections
  - Всеки обект от клас имплементиращ `__getitem__`
- Най-общо казано итераторите са контейнери за обекти, за да можем да “обходим” съответния обект.

Каква е разликата между итератор и итеруемо?

- итеруемото (iterable) притежава `__iter__` метода.
- итератора (iterator) притежава `__next__` метода.
- Използва се, за да се достъпи следващата стойност в итеруемото.
- Повдига грешката `StopIteration`, когато обхождането приключи

Многократно vs. еднократно

Многократно можем да итерираме списъци, множества, речници и тн.

```
squares = map(lambda x: x ** 2, range(5))
for number in squares:
   print(number)
# 0 1 4 9 16

for number in squares:
   print(number)
#
```

Обектите, които можем да итерираме еднократно, наричаме мързеливи. Това означава, че всеки елемент се генерира когато е необходим.

```
squares = map(lambda x: x ** 2, range(5))
print(list(squares))
# [0, 1, 4, 9, 16]
```

dunders

- `next(a)` **<=>** `a.__next__()`
- `iter(a)` **<=>** `a.__iter__()`

Обобщено за iter

- `iter` се опитва да извика `__iter__` метода на аргумента си
- ако се окаже, че такъв няма конструира итератор като просто извиква `__getitem__` с последователни естествени числа започвайки от нула,
- … докато не се хвърли грешка

Пример 1

```
nikulden = ['riba', 'bira', 'riba', 'bira']
nikulden_iterator = iter(nikulden)

print(next(nikulden_iterator))
```

Пример 2

```
class SequenceIterable:
    def __iter__(self):
        return SequenceIterator()

class SequenceIterator:
    def __init__(self):
        self.num = 0
    def __next__(self):
        value = self.num
        self.num += 3
        if value > 15:
            raise StopIteration
        return value
```

Пример 3

```
class Sequence:
    def __init__(self):
        self.num = 0
    def __iter__(self):
        return self
    def __next__(self):
        value = self.num
        self.num += 3
        if value > 15:
            raise StopIteration
        return value
```

Iter 2.0

Iter метода има и втора версия, в която въвеждаме втора стойност- така наречения стражник (sentinel). Итерацията ще спре, когато достигнем тази стойност.

Пример

```
class AddTwo:

    def __init__(self):
        self.start = 0

    def __iter__(self):
        return self

    def __next__(self):
        self.start += 2
        return self.start

    __call__ = __next__

my_iter = iter(AddTwo(), 8)

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

```
#2

#4

#6
```

Итераторите са мързеливи

```
example_list = [4, 2, 3, 1, 5]
example_list_iter = iter(example_list)
example_list[0] = 10
print(list(example_list_iter))
# [10, 2, 3, 1, 5]
```

reverse / reversed

- Каква е разликата между sort/sorted?
- sort променя оригиналния масив
- sorted връща нов сортиран масив
- Какво правят reverse / reversed?
- reversed не прави копия, а връща итератор

reverse / reversed (2)

```
numbers = [12, 15, 14, 10, 5, 7, 6]
reversed(numbers)
# <list_reverseiterator object at 0x7f14ff534490>
list(_)
# [6, 7, 5, 10, 14, 15, 12]
```

Генератори

- Генераторите са също итератори. Може би най-целесъобразния начин за създаване на итератор.
- Не пазят стойностите в паметта, а ги генерират когато е необходимо.
- Дефинират се като обикновена фунция с една основна разлика, вместо return използваме ключовата дума yield

yield

- Връща обект (итератор).
- `__iter__` и `__next__` се имплементират директно зад завесите.
- Когато функцията yield-не, не бива терминирана, а просто паузирана.
- Локалните променливи и техните текущи състояния са запомнени между последователни извиквания.
- Когато функцията приключи, `StopIteration` се повдига автоматично.

Пример с итератор

```
class SquaresUpTo:
    def __init__(self, up_to):
        self.up_to = up_to
        self.num = 0

    def __iter__(self):
        return self

    def __next__(self):
        if self.num > self.up_to:
            raise StopIteration

        square = self.num ** 2
        self.num += 1
        return square
```

Пример с генератор

```
def squares_up_to(number):
    value = 0
    while value <= number:
        yield value ** 2
        value += 1
```

list/set/dict comprehension

```
>>> [x ** 2 for x in range(5) if x % 2 == 0]
[0, 4, 16]
>>> {x: x ** 2 for x in range(5) if x % 2 == 0}
{0: 0, 2: 4, 4: 16}
>>> {x ** 2 for x in range(5) if x % 2 == 0}
{0, 16, 4}
```

Generator Expression

Като list comprehension, но с обли скоби и мързелив. :)

```
>>> example_generator = (letter * 5 for letter in "Bira")
>>> example_generator
<generator object <genexpr> at 0x00000162937f2650>
>>> list(example_generator)
['BBBBB', 'iiiii', 'rrrrr', 'aaaaa']
```

Функции по темата

- `any`, `all`
- `map`, `filter`
- `list`, `tuple`, `set`
- `enumerate`
- `zip`

map и filter

Приемат итеруеми и връщат итератори.

```
def numbers():
    num = 0
    while True:
        yield num
        num += 1

doubles = map(lambda num: num*2, numbers())
```

enumerate

```
necessities = ['Бира', 'Риба', 'Николай или Никола, или някое производно']
for index, necessity in enumerate(necessities, 1):
    print(f'{index}. {necessity}')

# 1. Бира
# 2. Риба
# 3. Николай или Никола, или някое производно
```

zip

```
from itertools import zip_longest
numbers = [1, 2, 3]
letters = ['a', 'b', 'c']
longest = range(5)
zipped_normal = zip(numbers, letters, longest)
zipped_longest = zip_longest(numbers, letters, longest, fillvalue='?')
print(zipped_normal)
# [(1, 'a', 0), (2, 'b', 1), (3, 'c', 2)]
print(zipped_longest)
[(1, 'a', 0), (2, 'b', 1), (3, 'c', 2), ('?', '?', 3), ('?', '?', 4)]
```

itertools

Разполага с всякакви видове функции за работа с итеруеми обекти. Всички функции в него са “мързеливи”

itertools.accumulate

```
from itertools import accumulate
sums = accumulate(range(1, 101), lambda a, b: a + b)
print(sums)
# <itertools.accumulate object at 0x1076d27c0>

next(sums)
# 1

next(sums)
# 3
```

itertools.combinations

```
from itertools import combinations
example = [1, 2, 3, 4]

combinations of two = combinations(example, 2)
print(list(combinations_of_two))

# [(1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4)]


print(list(combinations_of_two))


# []
```

itertools.cycle

```
from itertools import cycle
x = cycle([1,2,3])
for i in x:
   print(i)
# 1
# 2
# 3
# 1
# 2
.
.
.
```

itertools.count

```
>>> from itertools import count
>>> i = iter(count(5, 3))
>>> next(i)
5
>>> next(i)
8
>>> next(i)
11
>>> # ...
```

itertools.repeat

```
>>> from itertools import repeat
>>> list(repeat(10, 3))
[10, 10, 10]
```

Още itertools

- `itertools.repeat(objects[, times])`
  връща итеруемо с опредлен брой(или безкрайно много) повторения на един обект
- `itertools.cycle(iterable )`
  безкрайна конкатенация на един итеруем обект
- `itertools.filterfalse(function, iterable)`
  filter, тълкуващ предиката на обратно(ако function е None връща falsy елементите)
- `itertools.permutations(iterable)`
  генерира пермутациите на елементите в итеруемото
- `itertools.product(*iterables [,repeat=1])`
  връща декартово произведение на итеруеми
- `itertools.takewhile(function, iterable)`
  генерира елементите на итеруемото, до първото което не отговаря на предиката
- `itertools.dropwhile(function, iterable)`
  генерира елементите на итеруемото, от първото което не отговаря на предиката нататък
- `itertools.tee(iterable, n)`
  връща кортеж от n независими итеруеми

Разгледайте itertools

https://docs.python.org/3.10/library/itertools.html

Въпроси?